Security by Design : An asset-based approach to bridge the gap between architects and security experts

Nan Zhang Messe

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

Université Bretagne Sud

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ BRETAGNE SUD
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par
Nan MESSE

Security by Design : An asset-based approach to bridge the gap between architects and security experts. (Securité par la conception : Une approche basée sur les assets pour réduire le fossé entre les architectes et les experts de sécurité.)

Thèse présentée et soutenue à Vannes, le 7 janvier, 2021
Unité de recherche : IRISA (UMR CNRS 6047)
Thèse N° : 585

Rapporteurs avant soutenance :

Nicole Lévy	Professeur de l'Université au CNAM, CEDRIC
Benjamin NGUYEN	Professeur à l'INSA Centre Val de Loire, LIFO

Composition du Jury :

Président :	Yvon Kermarrec	Professeur à l'IMT Atlantique, Lab-STICC
Examinateurs :	Régine Laleau	Professeur à l'UPEC, LACL
	Xavier Le Bourdon	Dr. et Directeur de Recherche, DGA
	Jamal El Hachem	MCF, UBS, IRISA, Vannes.
Dir. de thèse :	Régis Fleurquin	MCF-HDR, UBS, IRISA, Vannes.
Co-dir. de thèse :	Nicolas Belloir	MCF, UBS, IRISA, Vannes, attaché aux Écoles de St-Cyr Coëtquidan

Invité(s) :
Salah Sadou Professeur à l'UBS, IRISA, Vannes.

Acknowledgement

Un simple merci n'est pas suffisant pour exprimer ma gratitude envers mon entourage qui m'a soutenue pendant ces trois années de thèse.

Je tiens tout d'abord à remercier la DGA et le pôle d'excellence cyber qui ont financé cette thèse.

Je tiens à remercier l'UBS, l'École doctoral Mathstic et l'IRISA de Vannes, qui m'a fourni un environnement de travail agréable.

Je tiens aussi à exprimer ma sincère gratitude aux membres de mon jury pour avoir bien voulu participer à l'évaluation de mes travaux de thèse et pour m'avoir donné des conseils très pertinents pour la suite de ma carrière. Je remercie spécialement mes deux rapporteurs Mme Nicole Lévy et M. Benjamin Nguyen pour leurs évaluations fortement appréciées. Je remercie particulièrement M. Yvon Kermarrec d'être venu assister à ma soutenance en présentiel dans ce contexte de crise sanitaire. Je suis heureuse de vous avoir rencontré en personne.

Ces trois années de vie à Vannes sont ancrées profondément dans mon cœur. Je ne les oublierai jamais. J'ai eu de la chance d'avoir une équipe très accueillante : Archware. Du fond de mon cœur, je remercie d'abord mes deux directeurs : Régis Fleurquin et Nicolas Belloir.

Régis : au début de notre rencontre, j'étais très impressionnée par ton niveau de rigueur et de connaissances, mais avec le temps qui passe, j'ai découvert également que sous cette apparence calme, il y a une grande bienveillance et de l'inquiétude pour les autres. Quand j'ai une question à laquelle je n'arrive pas à répondre, me faisant passer parfois plusieurs heures à chercher, tu arrives souvent, en utilisant quelques phrases concises et précises, à clarifier ma pensée. Jim et moi sommes tous les deux impressionnés par ta capacité de communication et d'écoute, la précision de ton vocabulaire et ta pédagogie. Tu es l'exemple que j'aimerais bien devenir en tant qu'enseignante-chercheuse. Tu as mon plus profond respect.

Nicolas : tu es la première personne que j'ai rencontrée à Vannes et avec qui j'ai échangé par mail pour l'obtention du financement de la thèse. Si j'aime la vie à Vannes et que j'ai aimé mes trois années de thèse, c'est en partie grâce à toi. J'apprécie beaucoup que tu

prennes grand soin de tes doctorants, non seulement pour leur travail, mais aussi pour leur santé et leur futur. Ton empathie pour les autres est inspirante. J'ai vraiment apprécié la fois où tu nous as fait sortir en mer avec ton bateau. Cela m'a aidée à soulager en partie mon esprit dans la situation difficile que l'on a vécue. Merci à toi de m'avoir donnée l'occasion de faire de l'enseignement aux écoles Saint-Cyr Coëtquidan. J'ai apprécié les discussions avec toi et Jérémy sur la route.

Ensuite, j'aimerais remercier Salah Sadou, Jamal El-Hachem et **Vanea Chiprianov**, qui ne sont pas mes encadrants officiels, mais qui se sont comportés comme tels.

Salah : nous avons passé de bons moments ensemble et bien ri. Tu aimes me charrier et j'aime te charrier en retour. L'ambiance de travail est agréable avec toi. Je te remercie d'avoir pris de ton temps pour ma thèse, même si je ne suis pas ta propre doctorante. Tu m'as toujours encouragée à publier dans des conférences de rang A et au final, cela a fonctionné. J'apprécie que tu m'aies donnée beaucoup d'opportunités pour faire de la recherche avec des chercheurs d'autres laboratoires. On n'a pas pu aller en Chine cette année, mais on peut essayer d'y faire une publication plus tard et je te guiderai.

Jamal : nous nous sommes rencontrées il y a peu de temps, mais tu es déjà une amie pour moi à qui je fais complétement confiance. Quand tu es arrivée à Vannes, tu étais un soleil qui a éclairci le laboratoire d'IRISA. Tu es tellement dynamique, souriante, et positive : c'est le caractère typique que j'adore. Je te considère vraiment comme une amie très proche après toutes les épreuves que l'on a passées ensemble. Je te souhaite du plus profond de mon cœur d'avoir plein de bonheur pour la suite et que ton cœur trouvera un apaisement.

Vanea : ta présence me manque énormément. Les doctorants qui t'ont eu comme encadrant ont vraiment de la chance et je suis heureuse d'avoir été parmi eux. Tu as été un encadrant idéal, ton départ a été une vraie perte pour nous tous.

J'aurais aimé...

J'aurais aimé partager avec toi la joie de l'acceptation du papier dans le TrustCom, ce papier que l'on a travaillé tous les jours avant que tu partes.

J'aurais aimé si tu pouvais être là pour ma soutenance, partager la réussite de ma thèse qui t'es en partie due.

J'aurais aimé que l'on continue les parties de jeux de plateau avec Jim qui te faisaient rire.

J'aurais aimé bénéficier de tes précieux conseils plus longtemps.

J'aimerais que tu sois encore là... J'espère que tu es encore là ... Et que tu reposes en

paix.

Je voudrais aussi remercier l'équipe DiverSE à l'IRISA Rennes, qui m'a accueillie chaleureusement pendant la fin de ma thèse en tant qu'ATER.

Je remercie aussi mes deux mentors qui m'ont beaucoup aidée à construire ma carrière pendant ces trois dernières années : Patrice Quinton et David Pichardie.

Je remercie également mes deux membres de CSI qui m'ont suivie pendant ces années et qui m'ont donné de très bons conseils : Jean-Michel Bruel et Gildas Menier.

Je remercie aussi tous mes stagiaires qui m'ont aidée pour ma thèse : Nicolas Blanchard, Etienne Descamps, Arnand Pernet, Haitam Sbaity, Justine L'helgoualc'h, Maiwenn Guillou, Pierre Goujon, Valentin Charier.

Je remercie aussi Pascale Launay, Pierre-Francois Marteau, Philippe Charton, qui m'ont laissé l'opportunité d'encadrer un projet de recherche, cela m'a permis la validation de ma thèse.

Je remercie également mes amis de Vannes : Lucie, Mathieu, Lionel, Laureline, Behzad, Paul, et tous les autres amis avec qui j'ai passé de très bons moments.

Je remercie ma famille en Chine : mes parents et ma sœur. Malgré la difficulté pour se voir à cause de la distance, vous êtes toujours mes soutiens.

Je remercie aussi ma famille en France : Anne, Daniel, et ma belle-famille. Je suis contente de vous avoir.

Finalement, je remercie la personne la plus importante pour moi : Jim, mon Lao Gong. J'ai de la chance de t'avoir rencontré. Tu m'as apporté amour, joie, sécurité, soutien, et tant d'autres. Tu es toujours là pour moi.

La vie à Vannes va me manquer et j'espère qu'un jour, je peux revenir ici pour continuer cette belle vie avec vous tous.

Table of Contents

List of Figures

List of Tables

General Introduction

Chapter 1

General Introduction

Contents

1.1 Context and problem statement

In today's digital age, software systems are interconnected and continuously communicating with each other, leading to higher security risks. With the increasing power and sophistication of attacks and rapid evolution of attackers' skills, serious financial losses have been caused. For example, according to Cybint [Cyb19], there is a cyber attack every 39 seconds, 43% of them target small-scale organizations, over 75% of healthcare facility has been infected by malware in 2018, and approximately $6 trillion is expected to be spent globally on cybersecurity by 2021. Accordingly, in the highly interconnected, ubiquitous computing world of today and tomorrow, cyber-security has become and will remain a major concern for organisations.

Modern software systems' development have mainly given priority to fast development processes, reducing significantly the time-to-market to adapt the modern development context and stakeholders' needs. However, a successful software-intensive system should not only deal with technical and functional aspects of software development, but also

satisfy non-functional properties, such as the security aspect. Unfortunately, in the majority of software projects, the correct security design and the assessment of security risks, aiming to properly configure and enforce security mechanisms, are often left out [Cas+20; Fou20]. Most of times, security is merely considered when the system has already been designed and put into operation [Mel+10], which gives the attackers the opportunity to cause security damages.

To deal with the security aspect, two strategies exist in modern secure software development: "*a priori*" and "*a posteriori*". Most of familiar methods focus on the "*a posteriori*" strategy, such as relying on secure infrastructure, setting up firewall and control access mechanisms, monitoring for intrusion, etc. "*A priori*" strategy, such as "security-by-design", has been paid less attention [VDB+18]. However, to avoid harmful damages with ever escalating attacks, designing security-aware software systems ("security-by-design") is essential [MFMP07; CHH16]. Traditionally considered as a low importance software property [Mel+10], security should become a major concern from the early development phases, because proactively integrating security into the software architecture and designing security-aware software systems have the potential of complying with security policies, leading to fewer security breaches, earning trust and reputation, and preventing a loss of business related to fixing security issues and thus avoiding future expenses in response to security breaches.

To allow the modeling of secure systems at the early development phases, security modeling languages, such as SecureUML [LBD02], UMLSec [Jür02] and SoSSecML [Hac+20], have been proposed to integrate security into the architecture design. In addition, risk assessment methodologies, such as NIST SP 800-30 [NIS12] and OCTAVE [AD02], also provide the possibility of integrating security aspects into the system development. Nevertheless, these methodologies are time-consuming (months or years) and require security expertise. Architects without significant security knowledge cannot effectively rely on these methodologies to deal with security aspects during the design phase. They need the minimum help from security experts. However, one of the key issues in the modern secure software development approaches is the lack and high cost of security experts [Cas+20].

In contrary, attackers are creative, they actively collaborate and they have powerful tools. For example, in 2017, a worldwide cyberattack by the WannaCry ransomware cryptoworm has occured. Around 200,000 computers were infected across 150 countries. Critical infrastructures in organizations such as hospital and transport domains were impacted. *Wannacry* worm propagated through the exploit *EternalBlue* that targets

Windows-based computers and exploits a vulnerability of the Server Message Block (SMB) protocol [Cen17]. One month later, another ransomware, *Petya*, has also used the SMB network spreading techniques to propagate among Windows-based computers even if Microsoft has released patches against *WannaCry* [Res17]. In 2019, *BlueKeep*, which is a vulnerability in Microsoft Remote Desktop Protocol (RDP) [Tec19], has been estimated to have the same disruptive potential as *EternalBlue*.

As we can see, with the growing need of security level for applications, services and technologies, the threats also grow as attackers get more organized and sophisticated. As the security defenses get better, attackers get smarter and can adapt their attacks to new defenses, and hence it is an on-going competition in cyber space. In the above examples, ransomwares emerge unceasingly with different attack techniques and/or attack forms on different specific targets such as SMB and RDP, but the common point of these attacks in the above examples is that they all target a vulnerable *network communication protocol*, which makes the propagation of malware to the whole network possible. To deal with this issue, studying attack goals in a more general and abstract level allows to not only analyze security aspects independently of specific domains or contexts, but also to understand the root cause of security breaches hidden behind various attack techniques and/or forms. It can thus help enhance the security level at the design phase.

1.1.1 Knowledge gap between security experts and architects: the architect' viewpoint

The "architects", being responsible of designing a software system, are knowledgeable about the system architecture design, but are not always well-trained in security [**lee2013**; San+17]. Thus they may not adequately deal with security aspects, such as *asset*, *vulnerability*, *attack* and *risk* concepts, when designing systems. However, architects are usually able to imagine a preliminary design solution, by adapting a "generic" solution to a "particular" context. For this, they can use either an *ad-hoc* Architecture Description Language (ADL) or a standardized generic one such as UML [OMG15] or SysML [OMG19].

To help architects conduct security-by-design, it is suggested to bring together in the design team several other participants, having different backgrounds and expertise than architects [Vli07], including security experts. However, involving security experts into the system design is not always possible due to the time-to-market and the budget constraints [Sho08; Cas+20], especially for small-scale companies. Even with the presence

of security experts when it is possible, architects can not always well understand the security jargon used by the security experts during their collaboration.

Therefore, there is a knowledge gap between architects and security experts [Fai+15]. The architects who have limited security knowledge thus need a **security assistance** to help them easily integrate security aspects into the architecture design, with or without the help of security experts.

1.1.2 Knowledge gap between security experts and architects: the security expert' viewpoint

To conduct security-by-design, one of the most important activities used in industries is the **threat modeling** [McG06]. It helps not only to address security issues at the design phase, but also is applicable at other software development phases, such as the implementation and the test phases. Conducting threat modeling at the early phases is more efficient than at other later phases because considering security aspects since the early development phases can help avoid costs and time spent on fixing security issues [OSC06].

Threat modeling aims at identifying a coverage of all possible threats [BHH13] and preventing and/or mitigating the effects of threats and attacks on a software system. Several threat modeling approaches exist, reviewed for example in [TÇS18; XR19]. As part of all these approaches, threat enumeration is at its core [Dhi11], which is traditionally carried out in brainstorming sessions. Current widespread threat modeling approaches (such as STRIDE [KG99], OCTAVE [Alb+03], PASTA [Uce12], etc.) are coarse-grained and require in-depth security knowledge. There is no detailed description of a procedure to support the brainstorming sessions, and no reference model to be used by such a procedure [Ysk+20; Sho14]. Due to the lack of guidance, the lack of sufficiently formalized process, the high dependence on participants (including security experts)' knowledge and the variety of participants' background, these brainstorming sessions are often conducted sub-optimally and require significant effort [FS09].

1.1.3 Security-by-design issues

In summary, we list the issues that can be encountered when conducting security-by-design, which are:

— The high variability of applicable threats over time, depending on the system deployment context and on platform and technical aspects;

— Security oriented software development life cycles are costly, time expensive and often need the involvement of security experts, which is nontrivial for small-scale organizations;

— There is a lack of a security assistance for architects, who have limited security knowledge, to easily integrate security aspects at the design phase;

— The brainstorming sessions in threat modeling are often conducted sub-optimally and require significant effort, due to the lack of guidance, the lack of sufficiently formalized process, the high dependence on participants (including security experts)' knowledge and the variety of participants' background;

— Security knowledge sharing is critical to popularize security aspects, but there is a lack of a shared terminology between security experts and non-security experts for the characterization of security. Security experts often use security jargons which are not always easily understandable by non-security experts.

1.2 Objectives and research questions

Modern software and system development urge novel methods and techniques [Cas+20] that:

— Enable to take security into account in each development iteration, possibly based on a security-by-design approach;

— Do not affect negatively the rapidity and flexibility of the development process;

— Are made of short, mostly automated tasks, which can be executed even by people with basic security skills.

Basing on these requirements, the work of this thesis focuses essentially on ensuring the security-by-design by bridging the gap between architects and security experts. The reason of bridging the gap between the software architecture world and the security world, is because it can attract and retain roles from the target research communities. The system needs to offer resources that are perceived as valuable, however these resources are also of value to a range of attackers [FS09]. The security needs must be met, or else they will not provide these resources. The security mechanism that protect resources must not be onerous to non-security experts.

Our aim in this thesis is to investigate how the improvement of threat modeling processes (bridging the gap from the security expert's viewpoint) can help the architects in integrating security aspects into the design phase (bridging the gap from the architect' viewpoint), thus helping to ensure the security-by-design.

To fulfill our general objective, we should answer the following research questions:

— **RQ1**: To bridge the gap between architects and security experts, how can we assist architects who have limited security knowledge to integrate security aspects at the design phase (from the architect' viewpoint)?

— **RQ2**: To bridge the gap between architects and security experts, how can we assist security experts to structure the threat modeling process to ensure the security-by-design (from security expert's viewpoint)?

1.3 Contributions

In this thesis, we aim at bridging the gap between architects and security experts by proposing a security-by-design approach that can be integrated within the most common methodologies, and we propose our contributions by answering the research questions RQ1 and RQ2.

To answer the RQ1, there is a need to

— Design a data model including the necessary concepts and the relations among these concepts to enable security-by-design;

— Propose a systematic process to guide architects integrating security aspects at the design phase.

For that, we propose the definition of a security assistance, which enables architects, who are not always security specialists, to integrate security aspects into the architecture model at the design phase. This security assistance allows to highlight known vulnerabilities contained in the architecture elements, threats that can exploit the discovered vulnerabilities, and to recommend security controls to architects if the suggested security controls are not already available in the target architecture.

This security assistance is based on an asset-based security assistance framework. This framework includes a data model that puts together the main aspects involved in the security characterization of a system and on the types of common elements involved

in a system, which enables to capture the most relevant aspects needed for the security analysis. This framework is also based on a systematic process to guide the security assistance.

Ideally, this security assistance allows to deal with partially abstract and partially concrete architectural models and with all types of Architecture Description Languages (ADL). The main objective of the security assistance in this thesis is to show that it is possible to define and automate such an assistance. The core of this security assistance promotes reworking on the concept of "asset" to help bridge the gap between the domain architecture expertise and security concerns, because *asset* is one of the concepts which is common to both architects and security experts.

To answer to RQ2, [Ysk+20] has identified the following research challenges, which are also our contributions in this thesis:

— Developing a reference model, which makes it possible to share threat modeling artifacts in a standardized manner for the reuse, the education, and the benchmark;

— Defining a systematic process that better supports the interactions among threat modeling participants, consequently, allowing a better knowledge reuse across projects, experts, and organizational boundaries.

For that, we propose a structured asset identification process, in order to improve current threat modeling processes. This process helps threat modeling participants collaboratively identify assets, which are significant for both business stakeholders and product team members, as well as for the security experts. It structures and identifies relevant assets, facilitating the threat enumeration during the brainstorming, by employing a number of concepts and relations, which we organise into a reference model. Moreover, to increase the knowledge reuse degree and reduce the reliance on subjective experience, we propose to construct a reusable vulnerable asset library as part of a threat library.

Accordingly, our contributions intend to reduce the human intervention by security experts and adapt the time-to-market constraint for the security-by-design. In summary, as shown in Figure 1.1, in this thesis we propose:

— A novel asset-based security assistance for the security-by-design that has several strengths. For example, this security assistance relies on a data model that includes the necessary concepts and the relations. Besides the data model, it is based on a BPMN-based process allowing the security assistance to be easily used by architects. In this way, it is able to support the security-by-design in multi-domains, taking continuously into account the security requirements in forms of security property

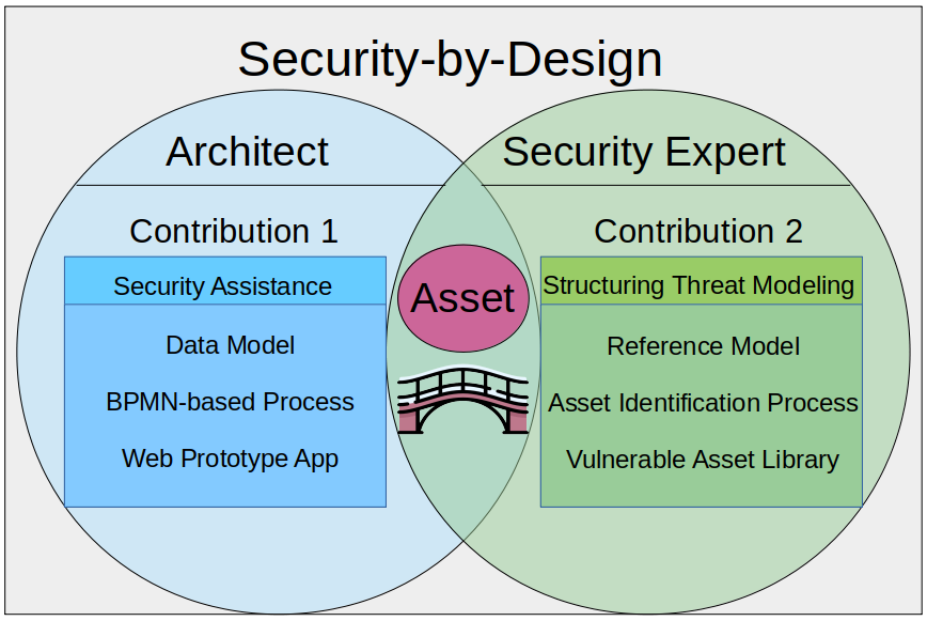


Figure 1.1 – Thesis Contributions to bridge the gap between architects and security experts

annotation. Moreover, it can be integrated within modern development processes that aim at solving the time-to-market constraint without altering the related application development flow. The proposed security assistance is implemented into a prototype web application tool that is used for presenting the feasibility of the tasks in the assistance process.

— A support to structure and formalize the threat modeling process, in order to bridge the gap from the security expert's viewpoint. This support relies on a reference model that includes the necessary concepts and the relations for the asset and threat identification. Besides, it is based on an asset identification process allowing to guide the threat modeling participants during the brainstorming. To allow the security knowledge reuse, we construct a reusable vulnerable asset library which can be included in the threat library. Furthermore, a prototype BASH application to extract vulnerable asset is introduced, which is extendable reusable by other security experts.

— We illustrate our proposed approach with real-world case studies and propose to evaluate our propositions with the crossover experimentation.

1.4 Thesis structure

The reminder of the thesis is structured as follows:

Part I Background and State-of-the-art: In Chapter 2, we call back the fundamental software engineering background and security background necessary in this thesis. In Chapter 3, the state of the art is presented and discussed with particular focuses on the integration of security aspects in the state-of-the-practice design phase, on the current threat modeling methodologies and process, and on approaches that are based on the concept of "asset", which we consider as the bridge.

Part II Contribution: In Chapter 4, we introduce our first contribution: bridging the gap between architects and security experts from the architect' viewpoint, by proposing a security assistance. This security assistance is included in an asset-based security assistance framework. This framework relies on a data model to preserve important concepts and relations, and a BPMN-based process, applicable on both single components of a software system and on the system as a whole. It is feasible to enact this security assistance. In Chapter 5, we present our second contribution to bridge the gap from the security experts' viewpoint, by structuring and improving the current threat modeling process. We present how an asset-based reference model can structure the asset identification phase, an essential phase in threat modeling. An asset identification process is also introduced to help improve the current threat modeling process, which in turn helps the security-by-design.

Part III Proof-of-concept: In Chapter 6, we evaluate our first contribution: the security assistance. We conduct an illustration on real-world case studies and propose to evaluate our propositions with the crossover experimentation. We also present a web application prototype tool that we have developed to allow the security assistance's enactment. In Chapter 7, we evaluate and discuss our second contribution: structuring the threat modeling. We illustrate the proposed methodology with case studies, and compare with another widely-used threat modeling process: Microsoft SDL threat modeling process. We show how our proposed asset identification process can be integrated into the Microsoft SDL threat modeling process, thus improving the current threat modeling process.

Part IV Conclusion: Finally, in Chapter 8, we conclude this thesis, we provide a final discussion of the presented approach and its limitations, and we present possible future works.

Part I

Background and State-of-the-art

Chapter 2

Background

Contents

2.1 Introduction

As we deal with "security-by-design" and aim at bridging the knowledge gap between the architecture design and the security worlds, we introduces in this chapter the necessary background knowledge required in this thesis. We divide it into two parts: the software engineering background (Section 2.2) and the security background (Section 2.3), to help both software engineers and security experts to have the minimum common vocabulary to understand each other.

2.2 Software engineering background

Software engineering concerns the development of complex software systems in a relatively long period of time, which involves a number of people with different expertise

and skills [Pre94]. Such development involves people such as software developers, testers, technical managers, general managers, stakeholders, etc. Each one has different knowledge background and different level of expertise about a subject or a domain.

In practice, especially in the modern digital age, the efficiency of the software development is of crucial importance for the quick delivery to satisfying the time-to-market constraint. The end-users' demand for new applications sometimes surpasses the workforce resource capacity. This gap between the supply and the demand is under growing with the increasing scale of the software system. To guarantee the efficiency of the large software system development, teams at different geographic locations have to cooperate. As we mentioned above, a number of people with different expertise and skills are involved. However, people management is a nontrivial task in software development. Coordinating and harmonizing the tasks of all participants and finding a right mix of skills for a development team are difficult matters. These problems quickly accumulate with the trend of distributed systems and globalization. To deal with these problems, it is suggested to bring together diverse participants in controlled ways in the software engineering field, basing on solid methods and techniques [Vli07].

2.2.1 Software development definition

According to the definition of IEEE Standard [IEE90], **software engineering** is the "*application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software*".

It begins with a solid theoretical basis and then uses robust and validated techniques to develop software systems [Pre94].

2.2.2 Software development in practice

In software engineering, a precise description of the development process can facilitate the understanding and communication for people who have to work together in the same team. Whether this description is in form of programming-language notation or a graphical notation, both support process management and improvement. Moreover, the software modeling process may serve as a basis for (semi-)automated tool support [Vli07]. As the software development is labor-intensive, the use of tools can help increase in productivity and thus safe the cost.

Besides the need of tool support in modern software development, software tends to be reused rather than to be produced for a more efficient and effective software development. Time-to-market, cost, and complexity constraints encourage organisations to assemble systems by reusing existing components, rather than developing those components from scratch. A software may extend from and/or interface with existing software, use available libraries, and build upon existing frameworks [JGJ97]. For this, software product line [PBL05] is used to conceive a set of software systems that share and reuse elements, which are tied to a given domain.

Another remark about the software development in practice is that one often uses more sophisticated process models. Recent software development methodologies, such as Agile [Coc06], are very popular and widely used. In agile methods, the design is more effective to be done as far as needed for the immediate next step, because the change is inevitable. Thus it is possible that the design in agile methods includes parts with different abstraction level. Moreover, agile methods are often people-oriented, which relies highly on the knowledge expertise of people involved. An effective collaborative and cooperative approach among all participants is thus essential in these sophisticated process models.

2.2.3 Software Development Life Cycle (SDLC)

Different methods and processes in software engineering field are proposed in literature, such as iterative model, waterfall model, spiral model, agile model, etc. However, we can always identify at least the following main phases among these models: requirement engineering, design, implementation, testing and maintenance [Som10; AB15].

— *The requirement engineering phase:* It focuses on eliciting, analyzing, defining, documenting, and maintaining stakeholders' requirements on the expected software system under consideration. It is a process of gathering and defining of what functions or services should be provided by the end product. During this phase, a complete description of the problem to be solved is introduced and the requirements are elicited together with its environment constraints. A requirement specification is documented during this phase. It can be served to validate the solutions proposed during the later phases that try to satisfy the elicited requirements [Som10].

— *The design phase:* At the design phase, a collection of hardware and software components and their interfaces to establish the framework for the software development are defined [IEE90], from both structural and behavioral aspects. It focuses on

solving the problems for the stakeholders and satisfying the elicited requirements by proposing solution models. The solution model can be decomposed into logical groups of functionality termed components, and into the interfaces that connect these components [Has18]. The design phase is often split into an early global architectural design phase and a later detailed one. Design decisions, captured in the architecture model, have a significant impact on the final product. It is worth noting that an available architectural description can be served as a template to other similar systems, as they may contain reusable components.

— The *implementation phase*: During this phase, software engineers concentrate on the individual components and on the composition of these components. The objective of this phase is to turn design decisions into codes, following the components' specifications in order to result in executable programs.

— The *testing phases*: It aims to detect errors or faults before the product delivery to decrease the cost as early as possible. There are two types of test during this phase: the verification (test the correctness of the transition between subsequent phases) and the validation (test whether the stakeholders' requirements are satisfied).

— The *maintenance phase*: It tries to repair the errors and the faults, and improve the software after its delivery. All type of changes (error correction, adaptation to user requests and environment changes, performance improvement, etc.) after the delivery of the software system are denoted by the maintenance.

It is worth noting that the above main phases may overlap. It is possible to start implementation of one part of the system while other parts are still under the design phase. It is also possible that the testing is carried out since the design phase. For the test, the earlier that errors are detected, the cheaper it costs to correct them [Som10]. For the maintenance, a substantial maintenance effort is inevitable since the context is unceasingly changing all the time.

The cost of developing the software and the cost of maintenance are of crucial importance in software engineering. Better methods and techniques for software development may result in larger financial savings [Pre94]. In order to decrease the cost and the effort, more attention should be paid at the early phases of software development such as the requirement engineering and the design phases.

The reason of the above claim is because that the decisions made at the early phases of software development have significant impact on the software product. As mentioned above, errors that are detected at the early phases are easier to be corrected than at the

later phases, thus resulting in large financial savings. [Boe87] considers that a successful software development should follow the "60–15–25" rule: 60% of investment in requirement engineering and design, 15% in implementation and 25% in testing. This is one of the reasons why our work concentrate on the early phases of software development, especially at the design phase.

2.3 Security background

With our increased dependence on software systems today, coupled with the rise of cyber-attackers on our digital information systems, cyber-security has become a major concern for organisations. It is imperative to proactively integrate security at the early phases of SDLC and to design security-aware software systems [MFMP07].

According to ISO 9126 [ISO01], **security** is defined as "the capability of the software product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them".

It is worth noting that this definition only covers the software product's security but not the one of process.

Nowadays, cyber security has become a matter of global interest and importance. The International Telecommunications Union (ITU) [(IT08] defines cyber security as follows:

Cybersecurity is the "collection of tools, policies, security concepts, security safeguards, guidelines, risk management approaches, actions, training, best practices, assurance and technologies that can be used to protect the cyber environment and organization and user's assets. Organization and user's assets include connected computing devices, personnel, infrastructure, applications, services, telecommunications systems, and the totality of transmitted and/or stored information in the cyber environment. Cybersecurity strives to ensure the attainment and maintenance of the security properties of the organization and user's assets against relevant security risks in the cyber environment".

According to [VSVN13], cyber security is not totally analogous to information security, although there is a substantial overlap between these two concepts. The main difference is that cyber security includes the human asset as potential targets of cyber attacks, whereas

in information security, reference to the human factor usually relates to the role(s) of humans in the security process.

There are a number of security fundamentals that underlie the development of secure systems. It is essential to understand the basic security concepts and the terminology used by the security community. In the following, we summarize some of the most important security concepts.

2.3.1 Security property

There are many security properties required or desired in secure software systems [Lee13a]. Three such properties have been called cornerstone security properties. They are *confidentiality, integrity and availability* (CIA). ISO/IEC 27000:2018 [ISO18] defines the *confidentiality* as "a property that information is not made available or disclosed to unauthorized individuals, entities, or processes". *Integrity* is "a property of accuracy and completeness". *Availability* is a "property of being accessible and usable on demand by an authorized entity". It is worth noting that *availability* is also an objective for reliable and fault-tolerant systems, except that *availability*, in the security sense, has to also consider the threats from malicious attackers, rather than just unintentional device failures [Lee13b].

There are several other desirable security properties other than CIA. For example, *access controls*, usually consisting of *authentication* and *authorization*, are essential components of controlling legitimate access to a protected object, which can be for example information, data, code or other resources of a computer system [Lee13b]. *Authentication* is defined as "a provision of assurance that a claimed characteristic of an entity is correct". Therefore, *authenticity* is a property that an entity is what it claims to be. Traditionally, *authenticity* can be ensured by one of the three methods: what you know (such as password), what you are (such as biometric) and what you have (such as security token or a private key). *Authorization* mechanisms determine what access the subject is allowed to the object. The entities involved in both two properties can refer to computers, devices, programs and humans. In our thesis, we only deal with CIA properties, the inclusion of other security properties is considered as the future work.

2.3.2 Elementary security concepts

Besides security properties, several other fundamental security concepts are also important. Here we review some common security concepts that are widely-used in security

engineering [SB14] methods. When we deal with security, we often deal with security controls, a security control mitigates at least an attack, and an attack exploits at least a vulnerability. Therefore, we consider *attack, vulnerability and security control* concepts as elementary concepts concerning the security.

— An *attack* is "an attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of an asset" [ISO18]. It is the realization or an instance of a *threat* that impacts computational resources. It involves detailed descriptions of the system, the vulnerabilities exploited, the attack path and the assets attacked [Lee13b]. "Know the enemy" is an important defense strategy in attack modeling, which means to understand the attackers' motivations, skills and their way of thinking, in order to defend successfully against attacks;

— A *vulnerability* is "a weakness of an asset that can be exploited by *attacks* to violate the system security policy and cause damage to an asset" [Gra+12]. Exploitable vulnerabilities expose an asset under a *risk*;

— A *security control*, defined by NIST [1] as safeguards or countermeasures, prescribed for an information system or an organization, designed to protect the confidentiality, integrity and availability of its information, and met a set of defined security requirements [NIS13], is a measure that modifies *risk*. A security control can be either a defense against a threat, which can be characterized as detection, prevention, mitigation, recovery and forensic defenses; or a countermeasure, which is a reactive security measures against an *attack*. Several security control frameworks have been proposed, both by standardization and industry-oriented organizations, such as the NIST Security Control Framework [NIS13], the ISO/IEC 27002 specification [ISO13a], etc. They list several security controls addressing different security domains and are related to both technical and organization aspects.

A security-by-design approach thus requires some forms of knowledge base that encodes at least *attack*, *vulnerability* and *security control* information.

2.3.3 Additional security concepts

Attack, vulnerability and security control notions are fundamental concepts in computer security [SB14], while other concepts, such as *asset*, *weakness*, *threat*, *security breach*, *countermeasure* and *security policy*, are also of significant importance and widely-used.

1. `https://www.nist.gov/`

— An *asset*, as defined by ISO 21827 [ISO08], is "anything that has value to an organisation", such as software, hardware, network, etc. As we can see, an asset can be anything valuable. This definition is vague. There are many different types and dimensions of assets that need to be protected against different levels of threats. For example, protecting a movie against illegitimate downloading has significantly different degree of importance than protecting national secrets against information excavation. The greater the risk is and the more valuable the asset is, the greater the degree of security protection is required [Lee13b];

— A *weakness*, not formalized explicitly in the ISO standard, has been highlighted by the non-profit Mitre Corporation [2]. Mitre defines weakness as a type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software. A vulnerability is an instance of an exploitation of a weakness;

— A *threat* is a potential cause of an unwanted security incident, which can result in harm to a system or organization [ISO18]. It is a class of attacks that violate one or more security properties, damage assets, or cause security breaches [Lee13b]. For example, Microsoft has developed STRIDE model to identify computer security threats. The six threat categories identified in STRIDE are *spoofing*, *tampering*, *repudiation*, *information disclosure*, *denial of service* and *elevation of privilege* [Sho14], which respectively threaten the security properties *authenticity*, *integrity*, non-*repudiability*, *confidentiality*, *availability* and *authorization*. A *threat agent* is the entity that gives rise to a threat and that concretely performs an attack against an asset to cause harm or damage. As we can see, an *attack* can be considered as a realization of a *threat*, hence we choose to focus only on *attack* concept in this thesis to represent an undesirable violation of a security policy;

— A *security breach* is an event that violates a security property. It is a consequence of a security attack. Examples of attacks that cause *confidentiality* breaches are eavesdropping, password cracking, information leakage, etc. Examples of attacks that cause *integrity* breaches are splicing attack, replay attack, corrupting audit logs, etc. Examples of attacks that cause *availability* breaches are Denial of Service (DoS) attack, distributed Denial of Service (DDoS) attack, network flooding, resource depletion, etc.;

2. The Mitre Corporation : `https://www.mitre.org/`

— A *countermeasure* or a *safeguard*, is prescribed for an information system or an organization to protect the *confidentiality*, *integrity*, and *availability* of the system and its information [RMO16]. By enforcing proper countermeasures, the potential damage of risks can be controlled;

— A security *policy* is a set of rules that governs all aspects of security-relevant system and system elements behavior. System elements include technology, machine, and human elements. The security policy rules can be stated at very high levels (e.g., an organizational policy defines acceptable behavior of employees in performing their mission/business functions) or at very low levels (e.g., an operating system policy that defines acceptable behavior of executing processes and use of resources by those processes) [RMO16]. A security policy is an intention and a direction of an organization, and is formally expressed by its top management;

— A security mechanism for the security community is a solution that involves enhancing existing design solutions to enforce the required security policies, thus defending against the threats. In the architecture community, it is called a "security architecture feature". Security mechanisms should ideally be part of the initial design or be integrated into the existing designs to ensure the security-by-design.

Figure 2.1 have synthesised and illustrated the above-mentioned security concepts involved in the security characterization of a system and their relations, merging together and extending different concepts introduced by several security standards and initiatives, including the Common Criteria for Information Technology Security Evaluation [DoD02], the ISO 27000 standard family [ISO18], the Common Weakness Enumeration (CWE) project [MITe], and other relevant works such as the one in [Cas+20].

2.3.4 Common existing security knowledge repositories

In practice, to identify concrete security information related to the above security concepts, such as attack exploit information, list of vulnerabilities that can be exploited, and mitigation information, security experts can rely on available security knowledge repositories. One of the advantages of these security repositories is that they allow the sharing of security knowledge, which is critical to popularize security aspects in secure software-intensive system development. One can employ the security knowledge repositories as much as possible in order to find related security information, such as attack patterns, vulnerabilities, weakness, security controls, and assets.

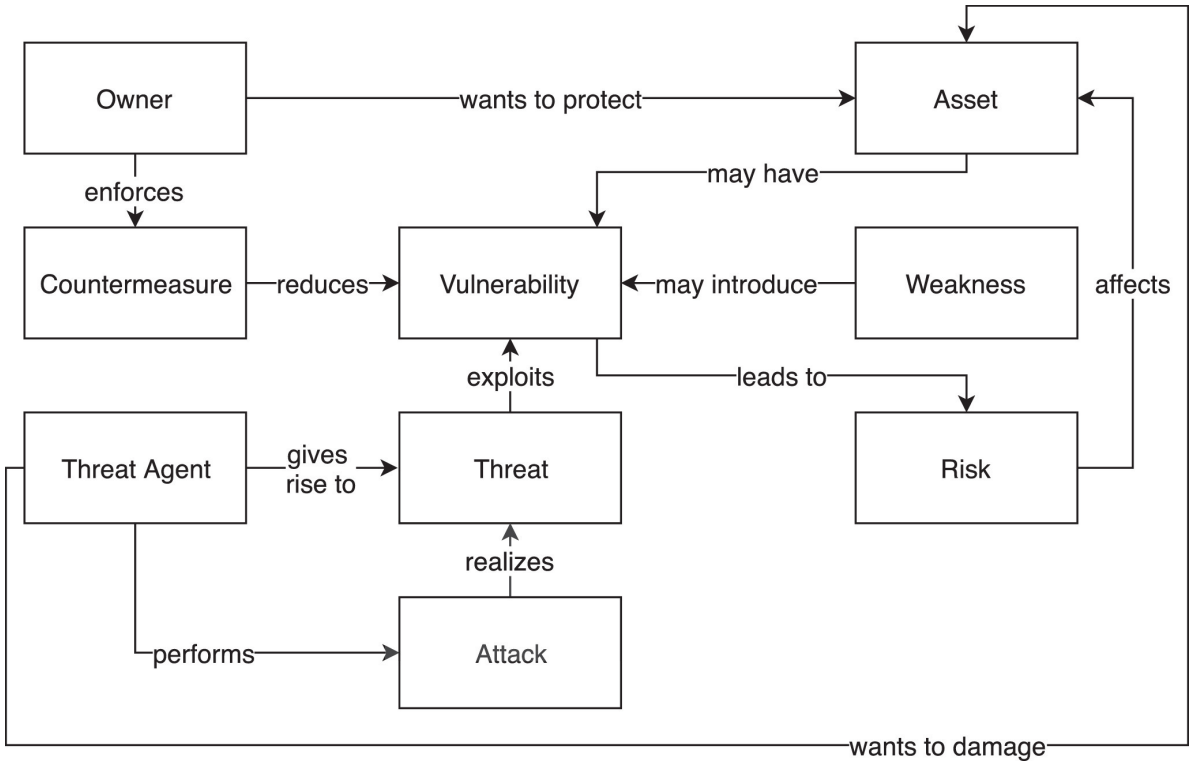


Figure 2.1 – Basic security concepts and their relations

In this section, three well-known public repositories are identified, which are Common Attack Pattern Enumeration and Classification (**CAPEC**) [MITb], Common Weakness Enumeration (**CWE**) [MITe] and Common Vulnerabilities and Explosures (**CVE**) [MITd]. There are other available security repositories such as OWASP [Fou17] and ATT&CK [MITa], which are not dealt with in this thesis for the moment.

CAPEC, CWE and CVE are all promoted by Mitre Corporation. MITRE has worked closely with the US government to strengthen cyber defenses for more than four decades, thus the public can rely on it to deal with the security aspects for general IT systems. Mitre Corporation offers the possibility to share and integrate security knowledge. It offers security information resources and defines a standardized way to refer to IT elements, via the Common Platform Enumeration (**CPE**) [MITc].

CAPEC, CWE and CVE provide the relevant security information, such as attack, vulnerability and security control information, by using the standardized representation of IT elements in CPE to present involved elements. In the following, we give more details about the information encoded by CAPEC, CWE and CVE.

Firstly, to build secure systems, it is important to "think like an attacker" and to discover vulnerable architecture elements, which can be considered as valuable assets for the attacker to cause damages. Accordingly, understanding the attacker's viewpoint is critical to build secure systems in order to prevent potential attacks. *Attack pattern* [MEL01] is a structured mechanism to capture and communicate this viewpoint. Attack pattern includes common approaches used by attackers to target software weaknesses. CAPEC provides a catalogue of common attack patterns that help understanding how attackers exploit weaknesses.

Initially released in 2007, CAPEC continues to evolve with public participation and contributions to form a structured mechanism for identifying, collecting, refining, and understanding attack patterns among the cyber security community. The attack patterns enumerated and classified by CAPEC are expressed in a hierarchical way.

Attack patterns in CAPEC are built from in-depth analysis of real-world exploit examples. It is also widely referenced by industries, governments, academia and standards, and supports a wide range of analytic use cases. The Telecommunication Standardization sector (ITU-T), a division of the International Telecommunication Union (ITU) responsible for coordinating standards for the telecommunications industry, approved recommendation X.1544 for the use of CAPEC as a standard in April 2013. ISO/IEC TR 20004:2015 also promotes the utilization of CAPEC.

CAPEC is based on graph views, which are basically hierarchical representations of attack patterns. The hierarchy of attack pattern follows this format: *Category - Meta - Standard - Detailed*. The first two levels (*Category and Meta* pattern) give abstract attack mechanisms, the last two levels (*Standard and Detailed* attack pattern) gather more concrete attacks.

CAPEC base helps to extract information about common security attacks because it appears to be the most complete repository composed of the largest number of attack patterns organized in an hierarchy providing various levels of abstraction. Attacks from the CAPEC repository can be extracted and organised into a tree format that describes a hierarchy of attacks from the most abstract to the most concrete ones so that all the sub-types of a given attack can be obtained.

Secondly, CWE collects common weaknesses that can be exploited by attack patterns in CAPEC. A weakness identified in CWE provides the possibility or the entry point for the attackers to realize attacks, which in a result can harm to IT systems. It is worth noting that weaknesses in CWE and attack patterns in CAPEC make references one to another through the sections called *Related Attack Patterns* and *Related Weaknesses* (e.g. CAPEC references CWE names and IDs in its *related weaknesses* field).

Security controls aim at reducing risks caused by vulnerabilities and at preventing attacks. Information about security controls can be retrieved from both CAPEC and CWE repositories from the "mitigation" field. Once an attack pattern or a weakness is identified, relevant mitigation can be retrieved to prevent the attack and control the weakness.

Another useful knowledge base is the CVE. CVE reflects real-world vulnerability exploitation and collects security incident data. Each vulnerability in CVE makes references with corresponding weakness in CWE. However, We don't deal with it in this thesis because much more data are neither reported nor revealed to the public, or waiting too long to inform the public by organisations, in order to keep their reputation [Var20]. Moreover, the vulnerabilities in CVE are platform- and technique-specific. Consequently, architects cannot only rely on such a public security breach repository to make security decisions, especially when part of architecture elements are still abstract.

Therefore, CAPEC and CWE can be helpful for our objective in this thesis, because they are based on the common feature of security breaches and/or the type of vulnerabilities, which is easily identifiable even for abstract architecture elements.

2.4 Conclusion

In this chapter, we have reminded the necessary background knowledge required in this thesis. We have reviewed the software engineering background and the security one. In software engineering background, we have mentioned its definition, its practice and the main phases in software development life cycle (SDLC). In security background, we have reviewed principle security properties, fundamental security concepts and common existing security knowledge repositories that can be reused. In the next chapter, we present the state of the art of "security-by-design" methodologies, and how existing works try to bridge the gap between security experts and non-security experts during their collaboration.

Chapter 3

State of the art

Contents

3.1 Introduction

In the Security Development Lifecycle (SDL) [HL06], two strategies are possible to integrate the security into the software development: "*a priori*" (integrating security at the early phases of SDLC) and "*a posteriori*" (integrating security at the later phases of SDLC). As we have already discussed, security should be a forethought in the SDLC, in order to avoid the degradation of performance, to save cost, to satisfy time-to-market constraint and to be ease-of-use. In recent years, more and more attention has been put

on the "*a priori*" strategy, even if security has been often considered "*a posteriori*" in practice [VDB+18].

In this chapter, we present the current state of the art regarding the "security-by-design" and "bridging the gap during the collaboration" research works, by reviewing related approaches that have been published up to this date. Firstly, Section 3.2 presents the current methodologies dedicated to integrate security into the early phases of software development, with a focus on the design phase. Secondly, in Section 3.3, we point out some current works about trade-off between security and usability, in the aim of bridging the gap between security experts and non-security experts during their collaboration.

3.2 Current security-by-design methodologies

Secure-by-design is under the scope of security engineering. Security engineering practices aim at building trustworthy and robust systems against possible attacks, threats and disruptions. These practices typically suggest the adoption of methodologies and processes that must be applied systematically to the target system and be carried out during its entire life cycle [RMO16], which is also termed Security Development Life Cycles (SDL), such as NIST 800-64 [Kis+08] and Cisco SDL [Cis]. These practices aim at validating the effectiveness of already enforced security controls, identifying existing weaknesses and guiding future security efforts and investments [EM10]. Current security engineering methodologies implicitly assume that: i) there is sufficient time to perform the secure software development (months or years) and ii) potentially a large number of actors, especially security experts are involved. These characteristics thus also apply to security-by-design methodologies.

We have mentioned before that large software-intense systems often reuse individual systems or components in order to be more efficient and effective, the functionalities from different component systems are often combined and harmonized to serve a common mission goal. One may wonder whether the security can also be compositional to deal with the time-consuming and people management constraints. The answer is negative, the security is not compositional as functional components, which means that the combination of the security solutions of component systems does not necessarily lead to a secure large composed system [Gar+10].

For example, the confidentiality level of the composed system has a "high water mark" effect. The composed system has the highest confidentiality level of any of its components'

confidentiality level (Bell La Padula model [EB11]). In contrary, the integrity level of the composed system has a "low water mark" effect. The composed system has the lowest integrity level of any of its components' integrity level (Biba model [Bib77]). Therefore, the security mechanisms of components with lower integrity level should be improved to preserve the high integrity level of the composed system. Moreover, the security level of interfaces connecting different components should also be studied for the security aspect. The earlier these security considerations are taken place, the less damage would be caused due to security breaches.

In the following, we firstly bring up current security-by-design methodologies together with their limitations in Section 3.2.1. Next, we focus on one of the most widely-used approaches that can support "security-by-design", the so-called "threat modeling" [McG06], which can help identify threats at the design phase. We present the state of the art of the threat modeling process in Section 3.2.2. Threat modeling integrates security in SDLC by identifying threats to and vulnerabilities of the system and by proposing mitigations. In threat modeling, the first step is to identify assets, which is a fundamental step. Therefore, in Section 3.2.3 we list some works that lay emphasis on the concept of "asset". We emphasize the reworking on the concept of "asset" because it is widely used in literature but vaguely defined, and we believe that it is the bridge between the architecture design and the security worlds.

3.2.1 Security-by-design methodologies

Designing secure software-intensive systems is an open challenge and various proposals try to address the issue by adapting Security Development Life Cycle (SDL) [MKM15; VDB+17].

The idea of "security-by-design" has been proposed in [CD13]. "Security-by-design" suggests "the adoption of proactive measures to deal with existing security threats, and it refers to a holistic, anticipatory approach based on a secure-by-default paradigm in the configuration of software components, access policies, and software security assurance processes". The "security-by-design" paradigm aims at identifying, as early as possible, potential security attacks and vulnerabilities, and designing secure components and interfaces since the beginning of SDLC [Cas+20].

More precisely, security-by-design approach requires integrating security at the design phase by adopting both software security assurance processes and trusted hardware [Cas+20]. Software assurance processes can be, for example, carrying out a compre-

hensive threat modeling and including the countermeasures against identified threats in the architecture. Trusted hardware can be, for example, tamper-proof hardware [TNP15], i.e., hardware designed to resist direct physical access adversaries. Trusted hardware often encompasses some cryptographic abilities, i.e., performing encryption and data authentication.

Moreover, since it is impossible to design a "one-size-fits-all" secure software-intensive system, it is essential to specify clearly the security requirements that are to be dealt with at the design phase.

We next present some of the major works devoting to security-by-design and we highlight their contributions and limitations. We choose to present these majors works under two main categories: model-driven security approaches and security patterns.

3.2.1.1 Model-driven security approaches

One possible way to ensure security-by-design is suggested by the model-driven security approaches [Ngu+15; VDB+17], because they can help generate technical security implementations from security requirement models. There are at least two ways in which model-driven engineering (MDE) and security might be beneficially combined: using MDE to support the development of secure systems and, integrating security techniques in MDE to give support to new development scenarios such as collaborative and distributed modeling.

Some model-based approaches cover risk assessment, attacker modeling and defensive architectures [Alp+19]. For example, Coras, Magerit and Mehari are model-based risk assessment methods. Formal models such as attack trees [MO06] and Petri nets [Pet77] model concrete attack paths on concrete assets, and they need the intervention of security expert.

Among the above identified methods, attack tree is widely-used in practice [SDP08; BFM04]. An attack tree [Sch99] is an hierarchical data structure that represents a set of potential techniques to exploit vulnerabilities. The security incident, which is the final attack goal, is represented as the root node of the tree, and the sub-goals that allow an attacker reaching the root are iteratively represented as branches and intermediate level nodes. Each node defines an action-based sub-goal, and each sub-goal may have its own set of further sub-goals. The bottom nodes on the paths, i.e., the leaf nodes, represent concrete actions to initiate an attack. Each node other than the leaf is either an AND-node (SAND-node) or an OR-node. To achieve the goal represented by an AND-node

(SAND-node), the sub-goals represented by all of that node's sub-nodes must be achieved (in a certain order); and for an OR-node, at least one of the sub-goals should be achieved.

As we can notice, the motivation of using attack trees is to effectively model how the attack goal can be compromised through different attack paths. However, to model attack paths, one has to have a minimum level of security knowledge, especially attack exploitation knowledge.

Besides model-based risk assessment methods and formal models, security architectures are usually described and analysed using modeling languages, such as UMLsec [Jür02] and SecureUML [LBD02]. Twenty-eight of these modeling languages are reviewed in [VDB+17]. This systematic review identifies, among other limitations, that only few (6 out of the 28 surveyed languages) propose automatic analysis mechanisms, which means that human intervention is inevitable for most of reviewed modeling languages.

UMLsec and SecureUML are the leading model-driven security notations that can be applied in generic domains and provide mature tool support. UMLSec is defined as a UML profile extension, using stereotypes, tagged values and constraints, to model security requirements and security mechanisms in order to satisfy security properties, such as confidentiality and integrity. Threat specifications correspond to actions taken by an adversary. Thus, different threat scenarios can also be specified based on adversary strengths. As to SecureUML, it defines the abstract syntax to annotate UML diagrams with the information pertaining to role-based access control [MD10]. While UMLsec is broadly applicable and covers a variety of security concerns, SecureUML is more specific and geared toward authorisation only. To model with UMLsec, one has to also have a minimum security expertise to specify threats and the corresponding actions taken by the adversary.

Other security-by-design works, such as the guidelines proposed in [Tum+19] to detect security design flaws, and four case studies discussed in [Cer+16] using a strategic, system-wide architectural approach with an implemented security framework, have also been proposed in literature. However, they lack a structured and formalized process to guide participants.

The landscape depicted by the above works outlines that the "security-by-design" goal could be difficult to be correctly attained in practice if the architects have little security knowledge and if no structured and formalized process has been proposed to guide the architects.

3.2.1.2 Security patterns

Security patterns, initially introduced in [YB97], are structured design patterns [Gam+94] to encapsulate and communicate proven and reusable security solutions to security problems. Security patterns introduce security into the development process [JR20]. These patterns can be expressed using UML class, sequence, state and activity diagrams [FB13].

Security patterns catalogs have been proposed in [FB13; Dou+09], such as patterns for identity management, authentication, access control, process management, etc. These security patterns are not directly related to vulnerabilities, but are directly related to threats, which means that pattern users have to foresee potential attacks about the system under design.

Although the evaluations generally point to the usefulness of patterns, pattern evaluation is currently the least explored topic by researchers and practitioners and there is a lack of empirical studies [JR20].

Similar to security patterns, [UF14] has combined the values of threat libraries and taxonomies, and has proposed a threat pattern, which is an extensible, two-level "pattern-based taxonomy" for (general) distributed systems. However, it cannot be easily used by those who have limited security knowledge.

3.2.1.3 Security-by-design limitations

To apply the above common secure-by-design approaches (whether model-driven security approaches or security patterns), the involvement of security experts is necessary, since non-security experts may have difficulties in identifying and understanding threats and vulnerabilities. However, one of the main limits of the current security-by-design approaches is the important financial cost: they assume the involvement of an expensive team of security experts over the whole development life cycle and/or of security-skilled developers. As a matter of fact, a few companies are actually applying the methodology as a common procedure, they apply it only for security-critical application development [Cas+20; Gee10]. For small-scale companies, security is often merely considered as an "additional requirement" for their products [Cas+20].

Current security-by-design methods are also time-consuming as they rely upon complex procedures that negatively affect the development process time line. The security design process and almost all phases of risk management life-cycle include at least the following main phases [NIS13]:

— The identification of the critical assets to be protected;

— The analysis of vulnerabilities of the identified assets;

— The threats that may exploit these vulnerabilities;

— The prioritization of risks relating to threats, vulnerabilities and assets;

— The selection and implementation of the security controls that are needed to protect the identified assets;

— The assessment of such security controls in order to verify that they are properly implemented, and their monitoring.

Sometimes, as in the agile method, the development time is short, one does not enforce strictly the secure design and the development standards within a limited time.

Another challenging problem for the security-by-design approach is that since modern software development projects have some rather peculiar characteristics (frequent changes in requirements during the development process, the rather invisible nature of the product during its development), there is a need for security-by-design procedures to be tailored towards software development and to adapt the changing context.

A more worrying concern is that the traditional risk-analysis techniques do not necessarily provide an easy guide (not to mention an exhaustive list) of how to identify potential vulnerabilities and threats at a certain component/environment level. This is why a large knowledge base, lots of security skills and experiences are invaluable. Therefore, security-by-design approaches are not yet widely well adopted in practice due to the lack of mature and guided solutions [Cas+20].

For small and medium enterprises and organizations who can not afford the financial cost for the involvement of security experts, architects should understand and master security issues to integrate security aspect at the design phase. However, security is not always well understood by the architecture community [Lee13b; MT18]. Architects designing secure systems must learn to think about how systems can be compromised or exploited by attackers, in order to design solutions proactively to prevent such malicious acts, which is an additional charge for architects.

Therefore, the existing approaches lack of an easily usable security assistance for architects, who have limited security knowledge, to integrate the security aspect into the architecture design (related to **RQ1**).

3.2.2 Threat modeling

To conduct security-by-design, one has to integrate the knowledge about threats, vulnerabilities and other security-related concepts at the design phase. Threat modeling is an effective approach to allow this integration. It is one of the most widely-used methods which is deeply integrated throughout the SDLC to identify security threats and to propose mitigations. Conducting threat modeling at the design phase is thus more efficient than at later phases [OSC06]. In this section, we focus on studying current methodologies and processes devoted to threat modeling, especially at the design phase. We firstly give the definition of the threat modeling and show its importance in secure software development. Then we make an inventory of the current threat modeling processes and highlight their limitations. Finally, we identify several needs or requirements that remain to be satisfied in threat modeling.

3.2.2.1 Threat modeling definition

Threat modeling is recognized as one of the most important activities in software security [McG06]. It aims at identifying a coverage of all possible threats [BHH13] and preventing and/or mitigating the effects of threats and attacks on a software system.

There are numerous definitions of threat modeling in literature, used in different and perhaps incompatible ways [XR19]. To synthesize, threat modeling is a systematic process of identifying and analyzing threats (i.e. potential attacks), which involves the understanding of threat agents' goals and actions in attacking a system, based on that system's assets [UF14; Bed+13].

Threat modeling is important and useful because it helps in [Ste10; TÇS18]:

— Identifying business-logic flaws and other critical vulnerabilities that expose core business assets;

— Enriching assessments with new potential attack vectors;

— Prioritizing the types of attacks to address;

— Mitigating the risks more effectively;

— Fixing issues early in the development process.

Threat modeling can occur at any time during the software development life cycle (requirements engineering, architecture/design, implementation, test/validation/verification and operation/maintenance), but it's more efficient to be performed during the early

requirement and the architecture/design phases [HL06], because fixing an issue that involves reworking on a conceptual model rather than significant re-engineering can save cost, development time and protect the system from high impact attacks [Tor05; Dhi11; Sim14].

Threat modeling is based on the identification of the system's valuable assets, such as sensitive information or the availability of certain processing facilities. Therefore, it can be applied at several levels of abstraction, depending on the type of assets considered [SWJ13]. It can also be supported by threat libraries or attack taxonomies [UF14].

Threat modeling is one of the software assurance processes that are widely applied in the industry. Several threat modeling methods have been proposed in the industry, such as STRIDE [KG99], OCTAVE [Alb+03], PASTA [Uce12], etc. In literature, several threat modeling methods have also been reviewed in [TÇS18; XR19]. For example, pwnPr3d [Joh+16] has been proposed as a probabilistic threat modeling approach for automatic attack graph generation that requires no security expertise. It is focusing on generating attack steps, which are mostly concrete steps and thus lacks an abstract level.

[TJR10] is another threat modeling method to provide security knowledge in the form of reusable security models and tool artifacts to help developers to build secure software systems. Their method requires to design and link threat models. A threat model is used to specify what assumptions are being made about the system, its users and the power of the attackers, the types of threats that the system defends against, and which threats are not considered. To design a threat model, one should have the necessary security knowledge.

[Kam16] has proposed to use threat classification models to analyze available threats basing on attack techniques and threat impacts. The proposed three models help in attacks mitigation process combining with STRIDE. However, they require a high level understanding of threat agents, their motivation, the frequency of attack occurrence and the estimated damage levels, which is a non-trivial task for participants with little security knowledge.

As part of all these threat modeling methods, threat enumeration is at its core [Dhi11; Ysk+20], which is traditionally carried out in brainstorming sessions. This activity depends heavily on the result of asset identification. However, there is no easy artifice/trick from assets to threats, no process that effectively guides and structures the interactions between the participants.

3.2.2.2 Threat modeling process

Phase	Asset Identification			Threat Enumeration				Threat Prioritization		Mitigation	
Activity / Paper	Identify security goal	Model domain	Identify asset	Identify threat	Enumerate &document threat	Describe attacker	Identify vulnerability	Rate threat	Assess risk	Mitigation	Verification
Torr (2005) [Tor05]		x		x						x	x
Shostack (2008) [Sho08]		x			x					x	x
Scandariato (2013) [SWJ13]		x		x	x						
Beckers (2013) [BHH13]		x	x	x	x	x					
Dhillon (2011) [Dhi11]		x		x					x	x	
Steven (2010) [Ste10]	x	x		x			x				
Kamatchi (2016) [Kam16]		x	x	x	x			x			

Table 3.1 – An inventory of threat modeling processes

Threat modeling process consists of several activities that deal with concepts of a threat model/pattern [UF14]. Threat modeling process is a collaborative process where participants include: business stakeholders; product-team members from all product development phases, such as enterprise, software and application architects, development leads, IT infrastructure specialists, engineers; security experts such as security analysts, security architects, threat modeling experts [Ste10; Tor05].

Several threat modeling processes including various activities are proposed in literature [BHH13; Kam16] and widely used in industry (Microsoft [Tor05; SWJ13], CIGITAL [Ste10] and EMC [Dhi11]), as shown in Table 3.1.

We summarize the threat modeling process into four main phases [Rhe+13; Dhi11]:

— *Asset Identification Phase*: It is centered on identifying security goals, modeling domains (by characterizing the system, usually by decomposing it and describing its components and data flows using (annotated) architecture/design diagrams) and identifying valuable assets;

— *Threat Enumeration Phase*: It is focused on identifying threats, together with attackers (their motivation and skill) and vulnerabilities, and enumerating and documenting resulted threats. This phase is often conducted in brainstorming meetings, sometimes guided by a threat library;

— *Threat Prioritization Phase*: It is based on the result of threat enumeration, to rate threats and assess risks. This phase can be either considered as an internal or external activity [TÇS18];

— *Mitigation Phase*: It aims at resolving threats by proposing security mitigations and by verifying them.

It is worth noting that not all the approaches in Table 3.1 include the asset identification activity, which is nonetheless a fundamental step. The activity of identifying asset is a bridge between domain modeling and threat identification, however, only a few works address this activity. Identifying assets is essential because it takes both into account the modeling of the domain under consideration, and the will of stakeholders to protect valuable elements. Without this step, the later threat enumeration and prioritization phases would be less efficient. Some approaches mention the activity of "identifying asset" [BHH13; Kam16], however, no detailed guidance or formalized process about how to systematically conduct this activity is proposed.

As a prerequisite for the threat enumeration (which is at the core of the threat modeling process [Dhi11; Ysk+20]), the quality of the asset identification impacts directly the threat enumeration and indirectly later phases. Therefore, the early phases (asset identification and threat enumeration) of threat modeling are crucial and are underpinnings for the success of threat modeling. However, the activities in these phases are often conducted in brainstorming sessions [Sho14], the results of which depend highly on the human expertise, experiences and collaboration, even with the support of widely-used methods such as STRIDE (a coarse-grained guiding method used largely in industry).

Brainstorming is the most traditional way to enumerate threats [Sho14]. The quality of the brainstorm is bounded by the experiences of the brainstormers and the amount of time spent brainstorming. As we can see, it is a subjective and unstructured activity, which needs a guidance that is more prescriptive, formal, reusable and less dependent on the aptitudes and knowledge of the participants.

For example, [Dhi11] has described EMC's real-world experiences with threat modeling focusing on the lessons learned by developers. The authors have found that developers achieve better results when guided by a threat library than when guided by STRIDE, because the threat library lightens the dependency on attack knowledge.

[Tor05] has demystified the threat modeling process in Microsoft. They have put an emphasis on holding a successful session of the threat brainstorming meeting, which should follow a methodical approach to enumerating threats, while still letting participants think about the problem creatively. There is thus a need to structure the brainstorming sessions while allowing participants being creative in identifying threats.

Therefore, conducting threat modeling requires ideally a sound knowledge of a sys-

tem's technical domain and sufficient security expertise to consider both generic and specific attacks for various system- and/or technology- specific contexts [UF14]. With the threat modeling process going on, more and more security expertise is required at each phase. However, the need of security knowledge can "leave most 'off-the-street' developers estranged" [UF14], leading to the result that threat modeling is performed sub-optimally or with significant effort involved, or not performed at all. Even after security training for participants, the threat modeling process is still difficult to execute [Ste10].

Therefore, threat modeling tasks are costly to perform, and require a high degree of collaboration among participants from different context and knowledge base.

3.2.2.3 Motivation for improving threat modeling process

As we have mentioned above, many of current threat modeling methods are coarse-grained and require in-depth security knowledge. There is no detailed description of a procedure to support the brainstorming sessions, and no reference model to be used by such a procedure [Ysk+20; Sho14]. Due to the lack of guidance, the lack of sufficiently formalized process, the high dependence on participants' knowledge and the variety of participants' background, these sessions are often conducted sub-optimally and require significant effort [FS09].

Threat modeling is therefore still at a low level of maturity [Ysk+20] and several key research challenges and/or requirements have been recently identified:

1) It is important to hold a successful brainstorming meeting [Sho14], which is still a subjective and unstructured activity. It should follow a methodical approach in enumerating threats, while still letting participants think about the problem creatively. It thus needs a guidance that is more prescriptive, formal, reusable and less dependent on the aptitudes and knowledge of the participants. Meanwhile, the cause of the high number of overlooked threats is also worthy of investigating [SWJ13].

2) The current threat modeling processes require a certain security knowledge level, making it a non-trivial task for participants with limited security knowledge. Proposed widespread threat modeling methods (such as STRIDE [KG99], OCTAVE [Alb+03], PASTA [Uce12], etc.) are abstract, coarse-grained and require in-depth security knowledge, for participants to apply it effectively to a specific product [Dhi11; Eke+06; TÇS18]. Inaccurate decisions are thus made based on insufficient knowledge about the security domain, threats, possible countermeasures and the own infrastructure. An in-depth reason is that security terminology is vaguely defined. This leads to confusion among experts as

well as the people who should be counseled and served [Don03]. There is thus a need to propose a method that can be easily used or understandable by security novices.

3) Moreover, a successful communication among threat modeling participants requires that they share their knowledge and points of view with as little bias and as few misunderstandings or confusion as possible [FS09]. Without a shared terminology communication, especially in a complicated domain like security, threat modeling cannot be successful [Eke+06]. Incorrect results could thus be caused by the misinterpretation of some template threats in the checklists. Therefore, there is a need of a common language or a common concept that can be understood by all participants.

Therefore, the existing threat modeling approaches lack of a structured and formalized asset identification process, and a common language and/or concept that can be easily understandable by security novices (related to **RQ2**).

3.2.3 The concept of "asset"

As the asset identification is an essential activity in threat modeling, thus also in security-by-design, we list some works that focus on the concept of "asset" and its identification in this section.

When we deal with attacks, we deal with the assets that have to be protected against the exploitation by attacks. The actual definition of "asset" advocates an eclectic view: "anything that has value to an organization". Assets that are potentially at risk are defined broadly, and may include information technology (IT) infrastructures, critical infrastructures (e.g., the power grid), intellectual property, financial data, service availability, productivity, sensitive information, personal information and reputation [Lee13b]. Some definitions are extensional, trying to enumerate types of system resources that are assets [SB14; Rau+16], which identifies for example: data, service provided by systems, system capability, such as processing power, item of system equipment, facility that houses system operations and equipment. In a word, asset can be anything that is valuable.

3.2.3.1 Asset-based risk assessment

The traditional process of risk assessment begins by listing valuable assets of an organization, then identifies threats to these assets and vulnerabilities exploitable by these threats, next estimates risks to these assets and finally proposes countermeasures to protect them. Therefore, *asset* is a central and pivotal concept. It is widely defined and used

in literature. However, most of risk analysts couldn't present a solid and sound asset analysis methodology until now.

Nevertheless, asset analysis is a critical step in risk assessment, involving the viewpoints of several actors, such as architects (representing the viewpoint of domain expert) and security experts. In traditional approaches, these actors work together to identify assets and use them for further risk assessment steps, such as threat enumeration, vulnerability identification and risk estimation. Therefore, the concept of *asset* implicitly contains the meanings with which each of these actors invest it.

Numerous risk assessment methods have used the asset notion, reviewed and compared for example by [Gri+18; SHS09], such as MEHARI, OCTAVE, IT-Grundschutz, MAGERIT, CRAMM, HTRA, NIST 800-30, RiskSafe Assessment, CORAS and Microsoft's Security Management Guide. For some of them, the concept of *asset* is defined very largely, rather vaguely, as anything that can have value to the organisation (NIST SP 800-30, CRAMM, ISO TR-13335, BS 7799 and OCTAVE). Other methods try to separate the *asset* concept into several types, such as EBIOS [ANS16] into primary and supporting, or as ISSRM into intangible business and tangible information system.

Moreover, [Tzi+16] has proposed an asset-driven, security-aware, service selection framework for selecting services that best satisfy the security and cost constraints of assets. However, none of the above methods, even if they deal with the security of the assets, identify types of assets from the attackers' point of view. Moreover, these risk assessment methods require long procedure times and the intervention of security experts.

Similarly, [Rau+16] has identified assets in the software architectural model, by mapping them from a system or organizational level. Their identification process is therefore focused on tracing assets from a development phase to another. The authors have also proposed a metric that quantifies software components by the assets they are able to access. However, they don't deal with assets form the security experts' viewpoints.

In [Cas+20], the author considers that a component is an asset and this component belongs to a component type. A component type can be associated with a threat condition. A threat condition models the set of conditions observed in a component behavior and internal implementation that may actually lead to the realization of a threat. For example, a web application component is exposed to "Cross-site Scripting" only in presence of "non-validated user-supplied data". The "non-validated user-supplied data" is a threat condition and the "Cross-site Scripting" is a threat for the component type "web application". However, the relation between a component type and a threat condition is

not formally formulated, because the threat condition is described in questionnaire-based natural language.

Similar to [Cas+20], the research in [Mes+19] has advanced the idea to refine the notion of "asset", but there is a lack of a formal representation of relevant important concepts and a systematic process.

[Moh+17] has mentioned that "asset" is anything valuable and needs to be protected. They consider asset as the "target of threats, the possessors of exposures, or the beneficiary of countermeasures", which considers the attackers' viewpoints. However, like [Cas+20] and [Mes+19], there is no structured and formalized process to identify this type of asset.

[MFMP07] has proposed a security resource repository meta-model to store all the reusable elements. The authors have added several concepts including "asset", based on the work of [SO03]. They have indicated that the asset can be valuable or critical, but also vulnerable. However, they have not provided details about how to systematically distinguish each type of asset. If an asset is not identified as valuable or critical at the beginning, but it is vulnerable, it can also cause severe consequence if the attacker succeeds to compromise this asset.

[BHH13] has proposed a structured method based on an attacker model to discover all threats, which is in compliance with common criteria [1]. The authors highlight the notion of "secondary asset", the harm of which can cause harm to a "primary asset". However, their work doesn't extend the chains of dependency relations among more than two related assets.

As we can notice, there is no solid and sound asset analysis methodology until the current date. To tackle this issue and cope with the system complexity, it is indispensable to reworking on the refinement of the notion of "asset" to represent different visions, from the architect, the attacker and the defender. It is also necessary to structure this novel refinement, in order to guide the participants in identifying assets. The notion of "asset" is worthy to be reworked because it has different meanings from different viewpoints, and all security controls, at the end, turn out to protect assets. Moreover, as each asset has its own owner, modeling assets can help to have a clear idea of the organization responsibility of each asset.

Therefore, proposing a lightweight method that can be easily used and learned by architects without extensive security knowledge, and an approach that bridges system domain specific architecture design with security attack and defense engineering (as sug-

1. https://www.commoncriteriaportal.org/cc/

gested for example by [JH12] and [Bod+09]) are thus necessary. For that, *asset* can be an entry point, which can be refined to present different viewpoints during the asset identification.

Therefore, the existing asset identification approaches lack of a structured and formalized asset identification process to bridge the gap between architects and security experts (related to both **RQ1** and **RQ2**).

3.3 Bridging the gap during the collaboration: the trade-off between security and usability

A software architect is a person who makes design and technological decisions in a software development project [McB07]. In practice, the software architect is not always a person or team dedicated to this activity, and in some organisations it does not exist as a differentiated role or as a job position [Ame+12]. Rather it may be performed by people having other duties as well, with many diverse tasks. Therefore, in practice, the architect may have different sets of knowledge, security included, but at different levels of proficiency. They may not be specialists in all these fields. While they are specialists in architecture design, their in-depth knowledge about non-functional aspects such as security can be very various. For example, they may have to learn attack trees and attack patterns to gain a deep understanding of attacks and countermeasures. This would require the architect a lot of effort, which is not practical.

A more suitable solution would be to enable the architect to discuss with the security expert who has all this in-depth knowledge. However, the communication can be not easy since they deal with different concepts and have different languages. For example, the architect talks about components and connectors [Has18], while the security expert talks about vulnerabilities, attacks and controls. For the architect to be able to discuss with the security expert, there is a need of a common language or a common concept that can be understood by both [FS09].

To help bridge the knowledge gap during their collaboration, several works have been proposed.

[Ste10] has proposed a process of relating threat modeling with architecture models. One of the first steps is to identify attack surfaces, but this step need a minimum level of security expertise.

[FS09] has used *anecdotes* and *scenarios* to express security knowledge and to reason

about security in order to facilitate the communication among different actors. *Anecdotes* are frequently used to communicate knowledge about real, concrete and specific security issues, whereas *scenarios* are even more widely used as a means of communicating security concepts, reasoning about security principles and justifying viewpoints. The identification of anecdotes and scenarios in secure system design has more practical and immediate implications for security-by-design in general. One of the key elements of any design exercise is to ensure the good communication among the participants.

Identification of asset can be used in conjunction with scenarios, which is a particularly useful method for explaining security concepts to security novices and reasoning. Identification of asset can be also used in conjunction with anecdotes which is a useful source of information that can be easily communicated, understood and accepted by different stakeholders. However, anecdotes used in security design discussions are not necessarily accurate or representative of the space problem. In practice, this needs the involvement of different stakeholders to discuss together. Structuring this process can facilitate the communication, which is not dealt with in their work.

Limitations of *anecdotes* and *scenarios* are related to the difficulty in generalising their information content. This means that *anecdotes* and *scenarios* contain highly specific descriptions of particular events in a system, whereas security needs have to encompass the system as a whole. Yet, this approach only deals with requirements models, it doesn't deal with domain models from other phases of the development lifecycle. Moreover, the security details involved are fine-grained and difficult to be generalised. Therefore, there is a lack of a reference model and a process that can be reused and can deal with multiple levels of security abstraction.

Besides, [Eke+06] has proposed a security ontology to resolve the communication problem. The authors have taken into account the entire infrastructure as asset which is physical and belongs to business domain. The ontology guarantees shared and accurate terminology in order to reduce misunderstandings. Their work aims at replacing the security expert but not proposing a collaborative process.

[Ame+12] has conducted 13 interviews with software architects. All the architects have declared that no specific tool were used for non-functional requirement (NFR) management. They are willing to accept some help in the form of a decision support tool to be assisted in the architectural decision-making or in suggesting alternatives. This study has shown that an assistance tool suite is demanded by architects to help them manage NFR such as the security requirement.

Therefore, the current works, aiming at bridging the gap during the collaboration, lack a reusable reference model and a structured process (related to **RQ1** and **RQ2**).

3.4 Conclusion

In this chapter, we have reviewed current approaches that have been proposed to deal with security-by-design and how to bridge the gap between security experts and non-security experts during the collaboration.

Most of the current security-by-design methods require the intervention of security experts, and are time-consuming (months or years). It is mainly costing to involve a security team, which is often troublesome to small-scale organizations. Even if security experts are included in the design team, the security jargons used by these security experts are not always easily understandable by architects who are security novices.

During their collaboration, the communication can be not easy since they deal with different concepts and have different languages. A shared terminology would be thus useful for the successful communication. We believe that "asset" is easily understandable by both architects and security experts, as they deal with "asset" in both architecture design and risk assessment.

However, there is a lack of a structured and formalized process to guide the architects and security experts in identifying assets, which degrades the result of threat enumeration in threat modeling. In practice, asset identification and threat enumeration activities are often conducted within brainstorming sessions. Due to the lack of guidance, the lack of sufficiently formalized process, the high dependence on participants (including security experts)' knowledge and the variety of participants' background, these brainstorming sessions are often conducted sub-optimally and require significant effort. The quality of the brainstorm is thus bounded by the experiences of the brainstormers and the amount of time spent. As we can see, the asset identification is a subjective and unstructured activity, which needs a guidance that is more prescriptive, formal, reusable and less dependent on the aptitudes and knowledge of the participants.

Therefore, a security assistance may be helpful at the design phase for those who are not competent in security to integrate security (**RQ1**), and the asset identification in threat modeling should be structured and be improved (**RQ2**) to help better collaboration between architects and security experts.

Part II

Contributions

Chapter 4

Contribution 1: Architect's viewpoint: A security assistance to bridge the knowledge gap between security experts and architects

Contents

4.1 Introduction

To answer the **RQ1** and counterbalance part of the limitations that have been addressed in Chapter 3, we introduce in this chapter our first contribution, which is to bridge the gap between architects and security experts, to help security-by-design from the architects' viewpoint. For that, we propose a security assistance.

More specifically, the proposed security assistance aims at coping with the challenges outlined in the analysis of the state of art in Chapter 3 by satisfying the following objectives:

— Simplify and reducing the effort from security experts and lowering security design costs, in order to make it affordable even for small-scaled organizations;

— The security assistance should identify vulnerabilities and threats concerning to architecture elements, and suggests to architects the relevant security controls;

— The security assistance should deal with both abstract and concrete architectural elements. Indeed, the architecture design can be performed in several stages and therefore it involves elements of different levels of abstraction. The architecture is refined gradually to become finally a concrete architecture model. It is thus possible to mix abstract elements (whose implementation details are not specific yet) with concrete ones (which are precise) in the same architecture model.

— The architect should be able to request the launch of the security assistance at any moment. The level of details and relevance of the security recommendations depend on the abstraction level of the architectural elements. The recommendations are more precise if the architecture elements are more concrete, referring to existing components, whose vulnerabilities are cataloged in common security knowledge repositories.

To enact this security assistance, it is indispensable to include the security knowledge into the assistance knowledge base, by adopting, as much as possible, standard solutions, available repositories and a standardized way to refer to IT products and platforms. The security assistance knowledge should at least include information about:

— *Attack pattern* (e.g. CAPEC) to represent the attackers' viewpoint, and to retrieve common attack goals and tactics;

— *Vulnerability or weakness* (e.g. CWE) that can be exploited by attackers;

— *Security control* that can be retrieved from mitigation information corresponding to attack pattern (e.g. CAPEC) and weakness (e.g. CWE);

— *Standard naming scheme* (e.g. CPE) to identify general IT elements in a standardized way and to allow automatic machine interpretation and processing.

At the current state of the art, none of approaches that we have reviewed in the Chapter 3 propose a security assistance integrating all the above knowledge.

In this chapter, we first introduce in Section 4.2 a motivating example to point out our motivation and to help illustrate the security assistance throughout the thesis. Then, we introduce the security assistance in Section 4.3. This assistance is based on a novel refinement of the concept "asset" to bridge the gap. This novel refinement of "asset" is included in an asset-based security assistance framework, which we present next. In the scope of this asset-based security assistance framework, we introduce an asset-based data model and a BPMN-based process, allowing the enactment of this security assistance to support security-by-design. Later, we conclude this chapter.

4.2 Motivating example

4.2.1 Motivating example description

To illustrate our approach throughout the thesis, we introduce a motivating example. It is a telemonitoring system for the heart failure management. This system leverages audio, video and other telecommunication technologies to enable the communication between the home care point, the hospital and the doctor's remote office. Heart Failure (HF) is a major public health concern, affecting at least 26 million people worldwide [SL17], resulting in more hospitalisations and higher health care costs among patients suffering HF [Cha+17]. Telemonitoring systems are used in order to reduce the burden of hospitalisations. Moreover, mobile phone-based telemonitoring systems along with application-based support of HF patient could decrease health expenditures [BF12].

As patients may undergo adverse events within the first months after suffering an acute HF [Sch+09], the development of such a system needs to be completed in a short time. An example of such a system is presented in Figure 4.1. In our example, we consider that the architect uses an ad-hoc Architecture Description Language (ADL) for the architecture

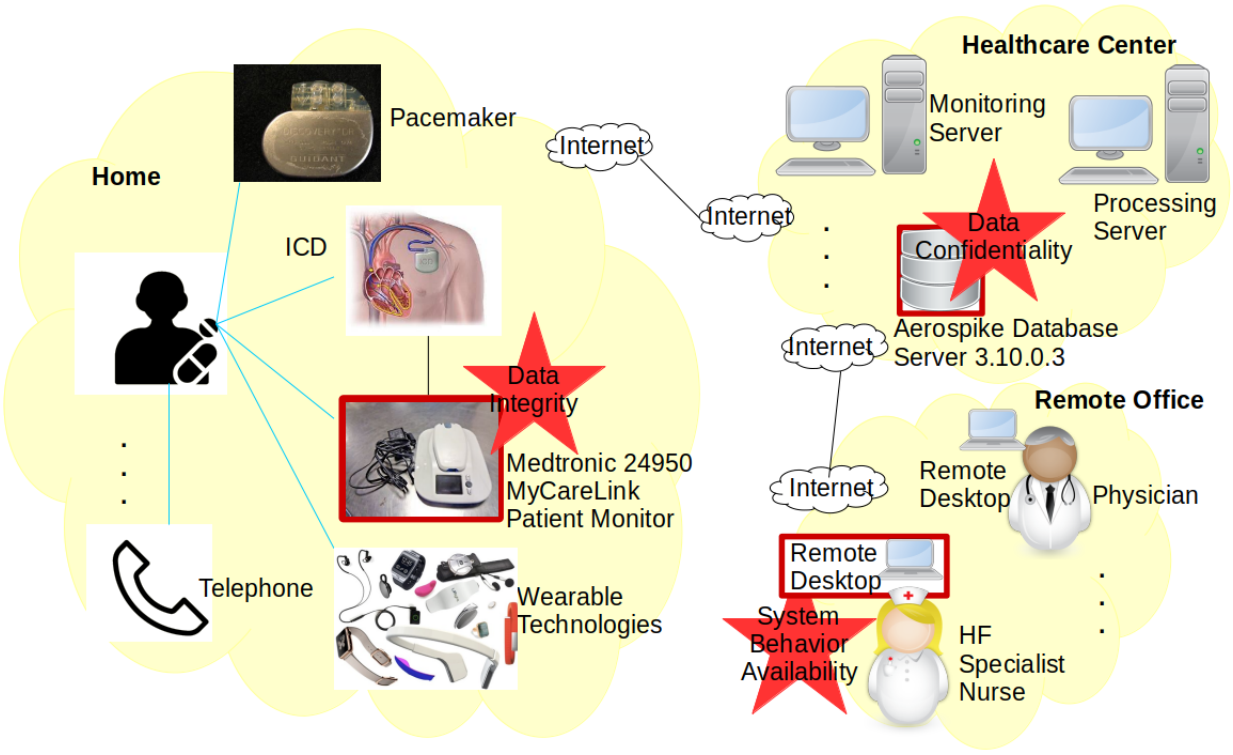


Figure 4.1 – Heart Failure Patient Health Telemonitoring System. Three annotated domain assets are identified: {Medtronic monitor, *integrity, data*} as annotated domain asset 1 (*DA1*), {Aerospike database server, *confidentiality, data*} as annotated domain asset 2 (*DA2*) and {remote desktop, *integrity, system behavior*} as annotated domain asset 3 (*DA3*). The "Medtronic 24950 MyCareLink patient monitor" and the "Aerospike Database Server 3.10.0.3" are concrete architecture elements while the "remote desktop" is an abstract architecture element.

modeling. As shown in Figure 4.1, the telemonitoring system is mainly composed of three parts:

— A home care point, for which we present with a subset of components such as a pacemaker, wearable technologies and a fix phone;

— A healthcare center with components such as monitoring and processing servers;

— A doctor's remote office with remote desktops.

This system involves a number of scenarios. In this section, we focus on the heart rhythm monitoring scenario to illustrate in the following how the security assistance helps to integrate the security aspect into the architecture design. In this scenario, the architect models an implantable device which is inside a patient's body, e.g. an ICD (Implantable

Cardioverter Defibrillator), to continuously monitor the heart rhythms and to automatically deliver therapies to correct fast heart rhythms when necessary. In addition, a monitor device (e.g. "Medtronic 24950 MyCareLink") is required to collect data from the ICD and transfer it to the healthcare center and doctor's remote office through the network, to help HF specialists make health decisions.

In reality, this simple scenario may be subject to a number of security attacks since there are no data and system behavior protection mechanisms in the architecture. For example, an attacker may tamper the data transferred from the "Medtronic monitor" to the healthcare center. For instance, heart failure signals could be modified into "normal" status signals. If a hearth failure happens, the specialist, who monitors the patient's health status in the remote office, should take decisions such as sending an ambulance. However, as the data would have been modified, the specialist could not take the correct decision of sending the ambulance, potentially leading to serious medical consequences and even the patient's death. Another example may be the Denial of Service (DoS) attack on the remote desktop in the doctor's remote office. If the patient suffers a hearth failure, even if the correct data is sent to the HF specialists, they cannot treat it in time because the system behavior and/or service of their desktop is not available.

4.2.2 Optimal security assistance scenario

In this example, the architect can not always rely on traditional security assessment methodologies to guarantee the security aspect due to limited time and to the possible absence of security experts. It is therefore difficult to integrate by himself/herself the security aspect at the design phase to protect the whole system from damages.

The general idea of our security assistance is that we take use of those which are understandable by architects, such as architecture elements (for example, a database server). According to the architecture elements and available security knowledge base, our assistance should identify their vulnerabilities and the threats which can exploit these vulnerabilities. For example, for the architecture element "database server", architects know that it contains and deals with SQL statements, but architects may not know that there is a potential threat of "SQL injection" if no input validation mechanism is set up. The objective of our security assistance is to relate these architecture elements with threats and vulnerabilities, to inform architects these threats, and to propose security controls, thus bridging the gap between architectural elements and security information from existing security repositories.

The trigger behind this security assistance is that architects have limited security skills in general [Lee13b; Cas+20]. However, to allow the security assistance, architects have to at least be able to:

— Distinguish the elements they want to protect (assets) in the architecture model;

— Be aware of some well-known security properties such as *confidentiality*, *integrity* and *availability*, which they want to preserve on these valuable assets.

Based on these information (which can be considered as security requirement) and the security knowledge databases, the security assistance returns alerts about the architectural elements' vulnerabilities and recommends countermeasures to prevent attacks that can exploit the vulnerabilities.

In Figure 4.1, some examples of architecture elements are highlighted in the red rectangles: two concrete architecture elements: i) "Medtronic 24950 MyCareLink patient monitor" (mentioned as "Medtronic monitor" later) and ii) "Aerospike database server 3.10.0.3"; iii) an abstract one "remote desktop". The architect chooses the "Medtronic monitor", which is at his/her disposal, as a concrete architecture element to play the monitoring role. Meanwhile, in the healthcare center, the "Aerospike database server 3.10.0.3" could be used to store the patient's data. In doctor's remote office, a "remote desktop", which the architect doesn't know yet more details about it, such as product name, vendor and version, is required to be allocated to the HF specialist to monitor the patient's condition. However, at this stage of the architecture modeling phase, the architect is not yet sure which concrete "remote desktop" to employ.

Secondly, the security assistance should enable the architect to indicate or annotate the security properties that need to be ensured for each chosen architecture elements (assets). For example, the architect may indicate the preservation of *Data Integrity* on the "Medtronic monitor", of *Data Confidentiality* on the "Aerospike Database Server 3.10.0.3" and of *System Behavior Availability* on the "remote desktop" (shown as stars in Figure 4.1).

To help ensure these properties, the assistance could interrogate its security knowledge base, about vulnerabilities that have been previously exploited by attacks, on similar architecture elements as those tagged as assets by the architect. If such vulnerabilities are discovered, alerts could be fed back together with possible security countermeasures. For example, for the *Data Integrity* of the "Medtronic monitor", a possible alert can be that there is a vulnerability of "using of hard-coded credentials", which may lead to serious attack consequences such as data tampering. A possible countermeasure "using a first

login mode" could be recommended to be taken into consideration during the design to change the default credentials in order to prevent the corresponding attacks.

4.3 Security assistance

In this section, we present how we design this security assistance to support security-by-design, with or without the involvement of security experts. As we have mentioned above, the security assistance should include information from available security repositories such as CAPEC and CWE. This security assistance should also relate the security information to architecture elements, which is a nontrivial task. To this aim, we propose a novel refinement of the "asset" concept, which we believe it as the bridge between architecture design and security worlds.

We firstly present this novel refinement of the "asset" concept. Then, we present an asset-based security assistance framework based on this novel refinement to represent the general view of this security assistance. Next, we introduce the "Attack Pivot Tree" (APT), which serves as the link between architecture elements and security concerns, by linking up different refinements of assets. After we elaborate an asset-based data model to structure the important concepts and their relations required for the security assistance. Finally, a BPMN-based security assistance process is proposed to illustrate how this security assistance can be enacted.

4.3.1 A novel refinement of the *asset* concept

As we have mentioned before, identifying asset is a fundamental step for a successful risk assessment. The asset analysis can involve the viewpoints of several actors, such as the architect, the attacker (tester) and the security expert (defender).

Our proposal is based on the *asset* concept. This concept can be easily understood by business stakeholders and by product team members [Rau+16; MFMP07]. It is also naturally well-known by security experts [ISO11]. It can therefore act as a shared concept between all participants from different backgrounds for the security-by-design.

There are numerous definitions of the *asset* concept in literature. For example, security engineering methodology ISO 21827 [ISO08] defines *asset* as anything that has value to the organisation, such as data, hardware, software or networks. Similar definitions focusing on the value of an asset, which can be subjective, commercial, and vary in a wide range,

are presented in [Rhe+13; Sho14; ISO13b; ANS16].

Several other definitions look at assets from the attackers' point of view, defining them as the things that an attacker tries to steal, modify, or disrupt, and considering their relations with threats [Tor05]. A few of definitions consider the relations of assets with vulnerabilities/exposures/weaknesses and countermeasures [Moh+17].

Even though, most of definitions remain too generic, too abstract and too wide-encompassing; entities that have various natures can all be included in these definitions. Such multiple overlapping definitions, including things attackers want, things stakeholders are protecting, and stepping stones, can "trip you up" [Sho14]. It may thus lead to misunderstandings and confusion among people from different background.

As *asset* means different things to different actors, such as the architect, the attacker (tester) and the security expert (defender), one usually requires a sound knowledge of a system's technical domain and sufficient security expertise to conduct a successful asset analysis in security-by-design [UF14].

In our opinion, *asset* encompasses three structural viewpoints:

— **The architect's viewpoint**, describes different architectural elements of the system under design, while focusing on the architect's understanding of the *asset* concept, such as software, hardware, network;

— **The attacker's viewpoint**, illustrates mainly a vulnerable architectural element as the target of an attack with the following information: attacker's tactic, exploitable vulnerabilities, security breaches and their impact;

— **The security defense expert's viewpoint**, who understands how the attacks are performed and proposes security control solutions such as vulnerability alerts and countermeasures to prevent attacks.

Based on this philosophy of differentiating different viewpoints, we next refine the concept of "asset" according to these viewpoints. This novel refinement is the foundation for the further presented data model and the BMPN-based process, which allow the security assistance integrating the attacker's viewpoint and the security defense expert's viewpoint with the architect's viewpoint, in order to help the architect to conduct security-by-design.

A same asset means differently under different viewpoints. For example, "Medtronic monitor" in the motivating example is a valuable asset from the architect's viewpoint, because it should perform its function as expected by stakeholders. Similarly, it is a valuable asset from the attacker's viewpoint, because it contains the vulnerabilities that the

attacker can exploit. A "Medtronic monitor" is also a valuable and vulnerable asset from the security defense expert's viewpoint because it is important and needs the protection against potential attacks.

We therefore refine the *asset* concept according to these three viewpoints to facilitate the fulfillment of the security assistance:

- **Domain Asset (DA):** Anything that has value to the architect and/or the domain expert, towards the fulfilment of the function and the goal of the system. It thus represents assets that are domain specific. For example, the "Aerospike database server 3.10.0" is a valuable domain asset to record patients' information.
- **Vulnerable Asset (VA):** Anything that has value to the attacker. It has vulnerabilities that can be exploited by attacks. Hence it is the direct, core target of the attacker. If it is compromised, it can impact related domain assets.
- **Vulnerable Domain Asset (VDA):** The domain assets that are also vulnerable assets constitute therefore a new type of asset, understandable by both the architects and security experts. We call this new type of asset **Vulnerable Domain Asset** (VDA). The VDA is therefore anything that has value for the architects, but also has vulnerabilities that can be threatened. As a VDA is a domain asset, it is thus domain specific. Our VDA concept has precursors in the literature. For example, [Sho14] remarks that the most common usage of asset in discussing threat models seems to be a marriage of "things attackers want" and "things you want to protect". However, in previous works, this idea is not further developed to show how to differentiate and structure different types of assets. As the VA operates at a high abstraction (pattern) level, we introduce the VDA as its projection (instance) on the architecture model. The security defense expert is aware of the value of the VDA for the architect and of its vulnerabilities exploitable by potential attacks and thus can propose relevant countermeasures.

Therefore, we consider the *asset* as the pivot and the bridge between domain architecture knowledge and security (attack and defense) aspects. Consequently, the originality of our approach is to bring together three different viewpoints from the architect, the attacker and the security defense expert, which helps us refine the concept of *asset*, into domain, vulnerable and domain vulnerable assets, as shown in Figure 4.2. DA and VA are all system artifacts but appear in a different context: respectively in a system architecture model and in an attack pattern description. However, they may also include elements which are exclusive to any one of them. As such, the domain assets may include assets

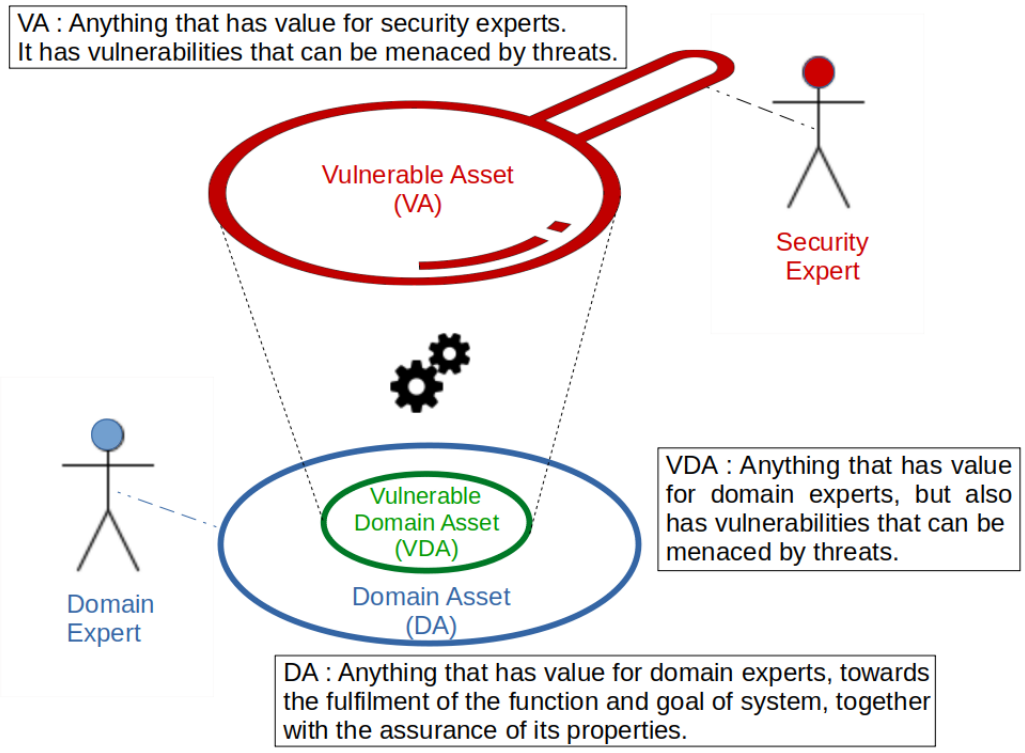


Figure 4.2 – General view of the novel asset refinement

which may not be vulnerable, and therefore not identified as vulnerable assets. Similarly, vulnerable assets may include assets which are not used in the architecture model. Determining the common assets between the domain and vulnerable assets is not trivial. However, it is essential, because they represent the domain assets that are also vulnerable and therefore need protection.

In contrast to domain assets, which are anchored to the architecture, vulnerable assets enable raising the security-based abstraction level. This allows tapping into the reusable attack knowledge patterns, which do not depend on the specific architecture model. As such, domain assets comprise architecture elements, which evolve with time, making domain assets evolutionary, varying and unstable, whereas vulnerable assets stay mostly the same and more stable because they are identified by common vulnerability types.

By distinguishing these two assets, the security assistance process is able to separate the asset which is the direct target of an attack (i.e., vulnerable asset) from secondary asset that suffers the consequences (i.e., domain asset), thus clarifying viewpoints from architects and security experts. For example, the “Aerospike database server 3.10.0” in

the motivating example is a vulnerable asset because it doesn't have the mechanism of testing the SQL input from users, attackers can thus perform SQL injection, while the "data" stored in this "database server" is a domain asset, which can be leaked if the direct target "database server" is compromised.

Moreover, the bridging concept "asset" also promotes the reusability of model elements using component libraries to store standard components such as specific network stacks, firewalls or operating systems [Joh+16]. Consequently, the originality of our approach is to bring together three viewpoints from the architects, the attackers and the security defense experts, by proposing a novel refinement on the concept of *asset*, in order to enact the security assistance, which is reusable in different domains.

4.3.2 Asset-based security assistance framework

We introduce an asset-based security assistance framework in Figure 4.3, to show the global view of the security assistance based on the above novel refinement of "asset". This framework allows the assistance integrating the security expert's viewpoint (attack and defense), in order to highlights vulnerable architectural elements and proposes security countermeasure recommendations to the architect to help architects to conduct security-by-design.

A security-by-design methodology supports the process of converting security requirements into a design that satisfies these security requirements, and can range from a set of guidelines to a predefined sequence of steps. Therefore, before beginning the security assistance, we have to note the security requirements that architects try to preserve.

From the architect's viewpoint as presented in Figure 4.3, an annotated domain asset is annotated on an architecture element to be protected (e.g. "Medtronic monitor", "Aerospike database server 3.10.0.3" and "remote desktop" in Figure 4.1), with a security property to be preserved (e.g., *confidentiality, integrity* and *availability*) and with a domain asset's category (e.g.,*data* and *system behavior*). An annotated domain asset can be considered as a security requirement.

In our motivating example in Figure 4.1, three annotated domain assets are identified: {Medtronic monitor, *integrity, data*} as domain asset 1 (*DA1*), {Aerospike database server, *confidentiality, data*} as domain asset 2 (*DA2*) and {remote desktop, *availability, system behavior*} as domain asset 3 (*DA3*). *DA1* signifies that the architect wants to protect the integrity of data in the Medtronic monitor, *DA2* means that the confidentiality of data in the Aerospike database server should be preserved and *DA3* signifies the system

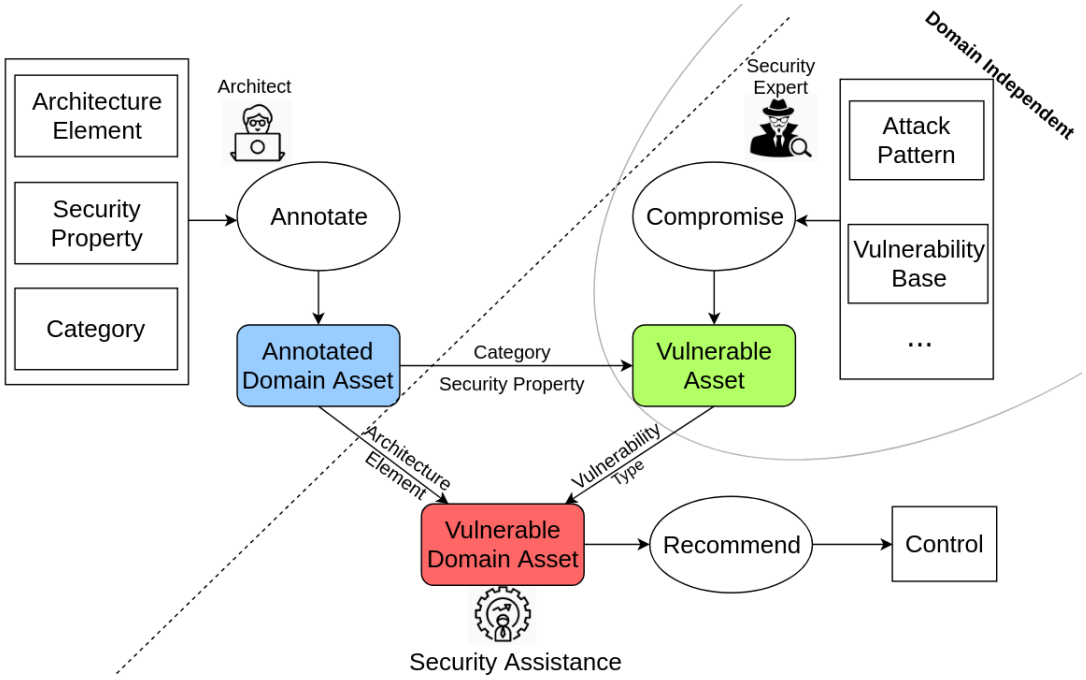


Figure 4.3 – Asset-Based Security Assistance Framework

behavior of the remote desktop should always be available.

From the security expert' viewpoint, the goal is to avoid that the attackers compromise a vulnerable asset using their knowledge of vulnerability types (e.g., contained in repositories such as CWE) and of attack patterns (e.g., captured in repositories like CAPEC). In contrast to domain assets, which concern with architecture elements, vulnerable assets enable raising the security-based abstraction level. They represent general direct attack targets, whose compromise can indirectly harm related domain assets. In our approach, vulnerable assets are retrieved from common security knowledge repositories (e.g. CAPEC and CWE).

Vulnerable assets are the core targets of attackers and if they are compromised, they can impact relevant *domain assets*, which involve architecture elements. As time goes by, new products which are potential architecture elements emerge and may suffer from various attacks, making domain assets unstable and vary with time passing, whereas vulnerable assets remain much more stable because they are identified by vulnerability types.

To define the mapping relation between DA and VA and to link them together, we rely on the concepts of *category* and *security property*. On the one hand, when compared

with DA, category and security property are model-independent (they do not contain any architecture element). On the other hand, the security property of a category can be impacted by the compromise of vulnerable assets.

In our motivating example, if the domain expert annotates the architecture element "Medtronic monitor" with the category *Data* and the security property *Integrity*, we obtain the "*Data Integrity* of Medtronic monitor" as an annotated domain asset, which can be considered as a security requirement. Using the category *Data* and the security property *Integrity*, our assistance approach searches in the integrated security knowledge base to find out the compromises of which vulnerable assets can have negative impacts on the *Data Integrity*. In our example, the assistance finds for example the VA "readable credentials". If "readable credentials" are obtained by the attacker, then remote services such as RDP, telnet, SSH, and VNC can be leveraged to log into a system, and malicious activities could be performed such as modifying the patient's heart rhythm, thus the *data integrity* is threatened. In this way, we can link DA with VA, which we give more details in Section 4.3.3.

Moreover, our security assistance takes into account the vulnerabilities of the assets that can be exploited by attacks. To protect the assets from being compromised, the assistance recommends security countermeasures. Therefore, it is also necessary to relate the concept of VDA with those of DA and VA.

If for a DA, the assistance finds at least one related VA, involving one or more vulnerability types, then this DA is presented as a VDA. In this way, our assistance uses domain-independent and general security attack knowledge to identify vulnerable architecture elements. For the "Medtronic monitor" (DA), it contains a "readable credential" (VA), thus the "Medtronic monitor's readable credential" is a VDA.

VA and VDA (together with other security knowledge such as vulnerabilities and controls) depict *assets* from the security expert's viewpoint. This information is usually not easily-understandable by architects. Therefore, our assistance encodes the knowledge of security expert's viewpoint, taking their roles and helping architects with security control recommendations, without their understanding of how perform attacks. In this way, the security assistance achieves this mission by bridging the three viewpoints relying on the novel refinement of the concept of *asset*.

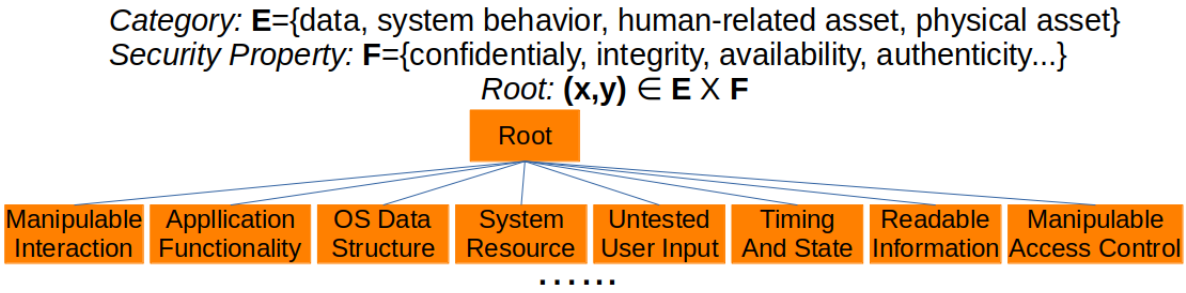


Figure 4.4 – The Attack Pivot Tree (APT)

4.3.3 Relation between Domain Asset and Vulnerable Asset via the Attack Pivot Tree

As we have mentioned above, an annotated DA valuable to the architect is related to an architecture element, a security property and a category. Our assistance needs to relate a DA to at least one VA in order to find a VDA. Attack Pivot Tree (APT) is used to allow this transition.

APT is designed based on the *Attack Tree* [Kum+18]. Our motivation of the use of APT is to effectively exploit the information available on attack patterns in CAPEC, and link them with architecture elements.

In an attack tree, the nodes are action-based, i.e., attack techniques. In comparison, APT is an asset-based attack tree, in which the nodes are either vulnerable assets or tactics, both of them are less technique-specific than the nodes in an attack tree, and have a certain level of abstraction. The VAs are the target of the attack tactics. Extracting the VA as an independent concept enables our assistance to make the connection with the DA from different contexts. Similarly to attack trees' nodes, the VAs of APTs can be refined and decomposed into more concrete and detailed ones.

Whereas VAs are inspired mainly from the two (of four) most abstract levels of attack patterns in CAPEC (*Category* and *Meta Attack Pattern*), attack tactic is inspired from the other two less abstract levels: *Standard* and *Detailed* Attack Pattern. A tactic in APT is an abstraction of the attack techniques in attack tree. Hence an APT is model-independent, whereas attack tree is used for specific domain architecture models. Similar to the techniques of an attack tree, which can be decomposed into more precise techniques, the tactics of an APT can be refined and decomposed into more concrete and detailed ones.

APT is designed to enable the link between DA and VA, the root of the APT is a

APT Vulnerable Asset	**CAPEC Attack Mechanism**	**Reference**
Manipulable interaction	Engage in deceptive interactions	CAPEC-156
Application functionality	Abuse existing functionality	CAPEC-210
OS data structure	Manipulate data structures	CAPEC-255
System resource	Manipulate system resources	CAPEC-262
Untested user input	Inject unexpected items	CAPEC-152
Timing and state	Manipulate timing and state	CAPEC-172
Readable information	Collect and analyze information	CAPEC-118
Manipulable access control	Subvert access control	CAPEC-125

Table 4.1 – VAs from Figure 4.4 extracted from CAPEC

special node. It constitutes the final goal of an APT, but at the same time it aggregates a category with a security property from an annotated DA. Figure 4.4 presents a short excerpt of the root and the first level of VAs of an APT. The root is a pair of two elements, the first element belonging to the set of possible categories, the second one belonging to the set of possible security properties. Each instance of the root (e.g. *data confidentiality*) is related to a subset of the VAs whose compromise can impact on the root.

The VAs are iteratively refined with each level of the APT. The most detailed level of the VAs are linked with tactics. A similar refinement exists for tactics. The top of the tree is the most abstract level while the leaves are the most concrete ones (respecting the four abstraction levels of CAPEC).

Both VAs and attack tactics can be related to vulnerabilities (extracted from the repository CWE) and to security controls, i.e., mitigations. In this way, the APT relates an architecture element of an annotated DA with one or several VAs, which are, in turn, related with vulnerabilities and security controls. The APT thus enables finding possible vulnerabilities and controls for an architecture element to assist the architect.

For example, in Figure 4.4, based on our security knowledge, we manually identify and extract the 8 children (VAs) of the root node (summarized in Table 4.1), from the most abstract level of the attack mechanisms in CAPEC.

As these eight mechanisms of attacks are the most abstract attack patterns in CAPEC, we follow the hierarchy of CAPEC to refine and decompose vulnerable assets that are extracted from attack patterns from different abstraction levels.

As we can notice, numerous concepts are used for the enactment of the security assistance until now. Therefore, according to the above philosophy, we next propose an approach which integrates and structures the necessary knowledge into a data model, presented in Section 4.3.4. We also propose a sequence of tasks organised in a BPMN-based process according to this data model to enact the security assistance (Section 4.3.5).

4.3.4 Security assistance data model

In order to structure important concepts and enact the security assistance, we have designed a data model enriching the theoretical model reported in Figure 2.1, with additional information related to assets, threats and countermeasures. This data model is depicted in Figure 4.5 (which is split into four parts later for the simpler readability), extending existing security standards and including the security concepts that enable a guided identification (i) of the vulnerable domain assets, (ii) of the vulnerability types of these vulnerable domain assets and the attacks that can exploit them, and (iii) of the countermeasures needed to mitigate such threats.

The data model is built based on the novel refinement of the concept asset presented in Section 4.3.1, which specialize generic assets based on the considered architecture model and that are directly mapped to the vulnerable assets. Basically, we associate element types (e.g., web applications, storage services, etc.) with a set of well-known threats that potentially affect all elements or components of that type (e.g., threat "Cross-site scripting" potentially affects "web application" element type and threat "SQL injection" potentially affects "database server" element type).

For the simpler readability, we split the data model into four parts, according to the different viewpoints, represented with different colors: the domain architecture (blue), the security knowledge including the attack (green) and the defense (red) parts. The fourth transverse part describes the structure and refinement mechanisms of concepts presented in the different viewpoints. To ensure that the relations between these four parts are preserved, some concepts are reused in several parts, such as *Root*, *VulnerableAsset*, *VulnerabilityType*, *ArchitectureElement*, etc.

4.3.4.1 Domain architecture specific aspects

The concepts described in Figure 4.6 capture the specificities of the domain architecture under study (e.g., healthcare), which are required as the entry point to launch the

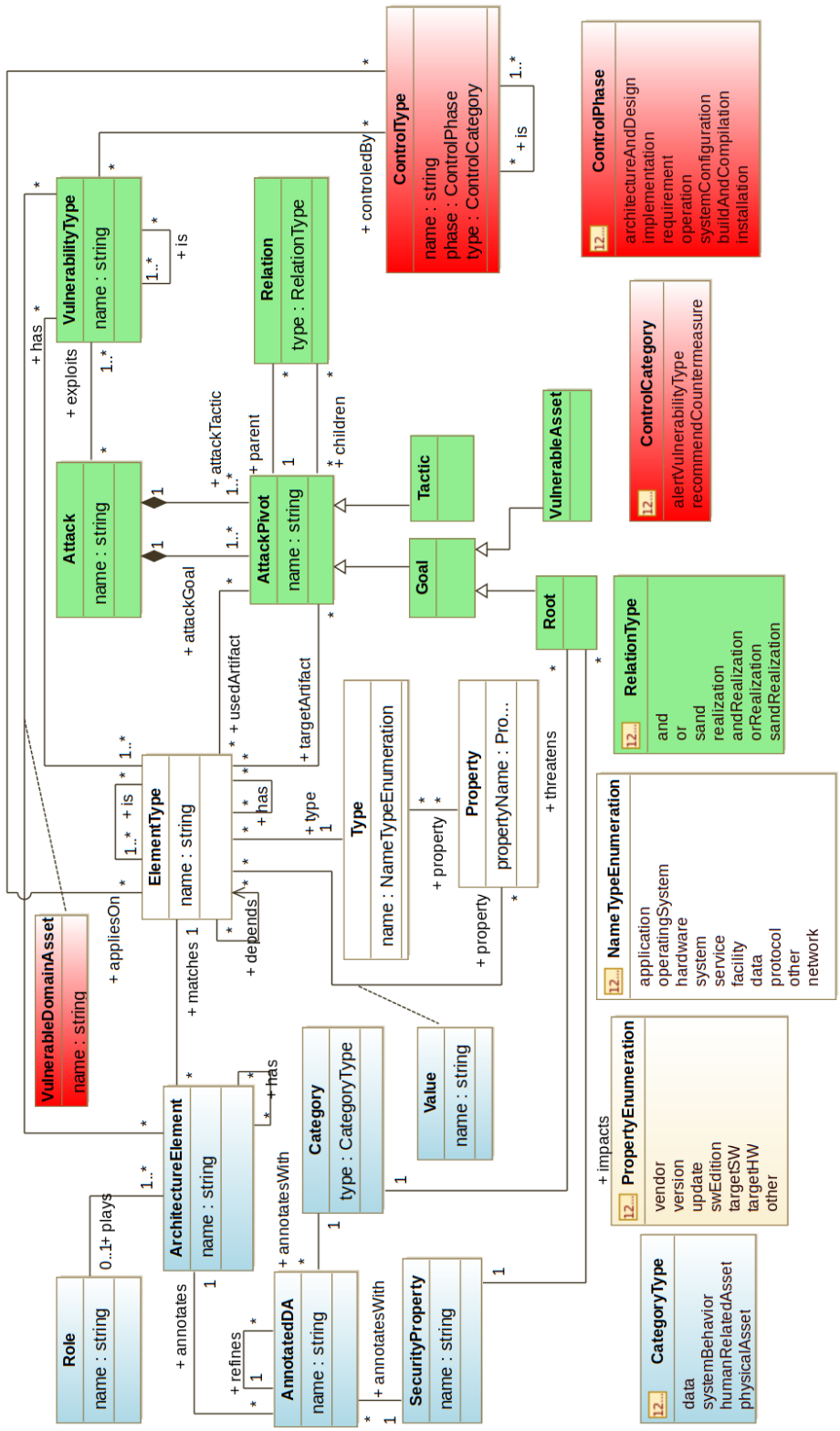


Figure 4.5 – Security Assistance Data Model (It is split into four parts below for the simpler readability)

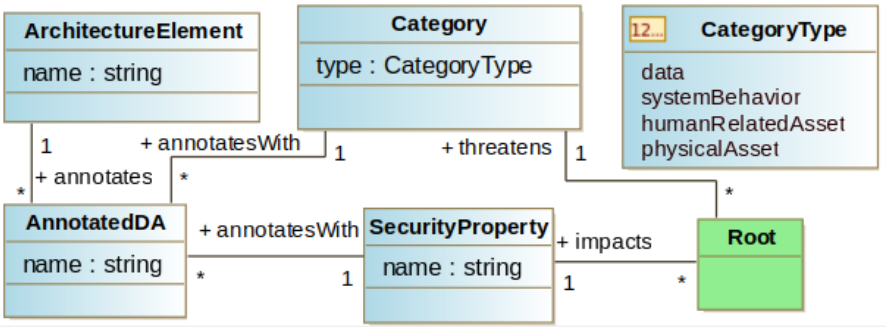


Figure 4.6 – Domain Specific Architecture

security assistance.

An *AnnotatedDA* (cf. definition in Section 4.3.1) annotates a domain asset, which is an *ArchitectureElement*, annotated with a *SecurityProperty* and a *Category*. As presented previously, the *SecurityProperty* and *Category* are used to relate the *AnnotatedDA* to the *VulnerableAsset*, through the concept of *Root* in an APT. The compromise of the *Root*, in its characteristic as the final goal of the APT (cf. Section 4.3.3), *threatens* a *Category* and *impacts* a *SecurityProperty*.

An *ArchitectureElement* is an element represented in the architecture model. A *SecurityProperty* expresses the security objective that the architect wants to protect. Here we consider the most studied ones: confidentiality, integrity and availability [TBB18; Mal+19; ISO18]. The architect is supposed to understand these properties and annotates them on the *ArchitectureElement* to require the preservation of these properties.

Systems often encompass more than software and hardware. Other materials, humans, work practices, belong to the system as well [Vli07]. Human resources are regarded as a critical resource, thus an asset as well. A *Category* represents the category of the *AnnotatedDA* to be protected. Based on assets that are commonly impacted by security attack consequences, we have identified four types of categories. Indeed, most of threats in [SB14] have consequences on data and/or system behavior. Moreover, as people can be involved, attacks may have possible consequence on human-related assets, e.g. user identity. Finally, the destruction of physical devices is also a possible consequence. Hence *CategoryType* contains: *data*, *systemBehavior*, *humanRelated* and *physicalAsset*. They synthesize most of potential impacts of attacks.

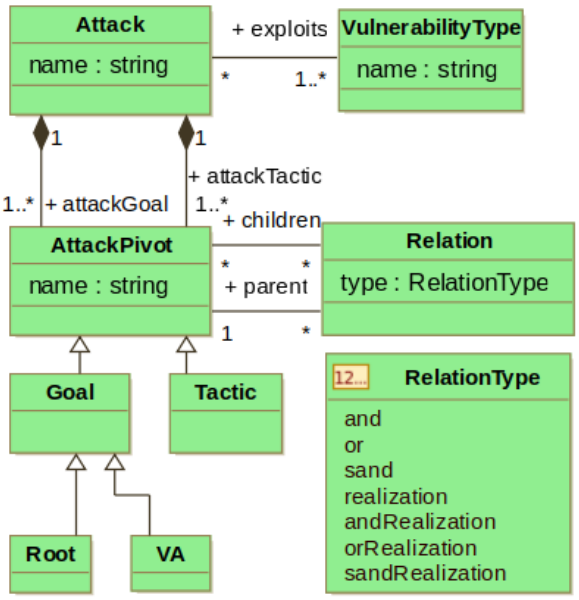


Figure 4.7 – Attack Specific Aspects

4.3.4.2 Attack specific aspects

This subsection captures the attacker's viewpoint as shown in Figure 4.7. The *AttackPivot* is inspired from the concept of *node* of an *attack tree*. It represents at once the *Goal* and the *Tactic*. As discussed, we differentiate the *Goals* into *Root* and *VA* (cf. Section 4.3.1). As these nodes are centered on *asset*, which is pivotal to our approach, we name these nodes: *AttackPivot*. The *AttackPivots* are related with each other through *Relation* of the type *RelationType*: and, or, sand (sequential and). A *Goal* relates with one or several *Tactics* through *and/or/sand-realization* relations. The *RelationType realization* means that a *Tactic* can realize the compromise of a *VA*.

An *Attack* is composed of at least a *Goal* and a *Tactic*, which constitutes the attack means to compromise the attack goal. It also consists of exploits performed by an attacker, to take advantage of *VulnerabilityTypes* to cause negative impacts. A *VulnerabilityType* models a vulnerability, a weakness or a design error that may result in an undesirable event, whose exploitation can compromise the involved *VA*.

4.3.4.3 Defense specific aspects

The concepts discussed in this subsection capture the viewpoint of the security assistance (defense) as shown in Figure 4.8. An *ArchitectureElement* can be matched to

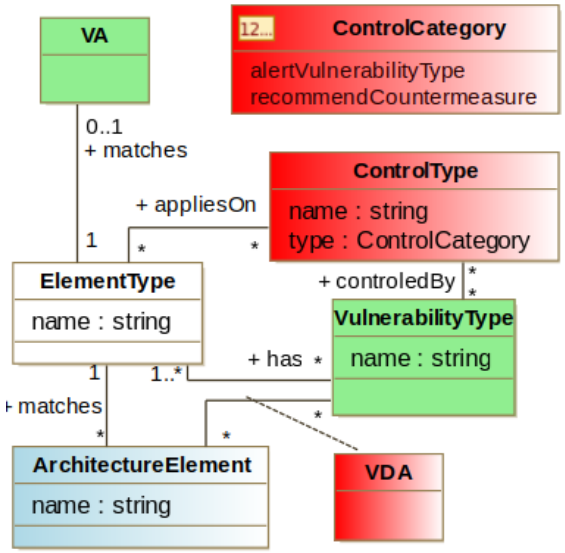


Figure 4.8 – Defense Specific Aspects

a general *ElementType* to pass from domain-dependent to domain-independent. If an *ElementType has* at least one *VulnerabilityType*, the corresponding *ArchitectureElement* becomes a *VDA* (cf. Section 4.3.1). To mitigate the *VulnerabilityType*, *ControlTypes applying on ElementTypes* may be proposed. These *ControlTypes* may fall in one of the two *ControlCategories*: *alerts* about the *VulnerabilityTypes* whose exploitation may impact the *ElementTypes*, or *recommendations of countermeasures* to prevent attacks. A noteworthy remark is related to *ElementType* being connected to concepts from all three viewpoints. As such, an *ArchitectureElement* from the architect's viewpoint *matches* an *ElementType*. An *ElementType having VulnerabilityTypes* may *match* a *VA*, from the attacker's viewpoint. A *ControlType applies on* an *ElementType* from the defender's viewpoint. The concept of *ElementType* encodes information about possible types of elements in an architecture. Our security assistance knowledge base contains a list of such *ElementTypes* and their relations. This enacts the assistance to make the link between domain specific knowledge and security attack knowledge through the *match* relations with *ArchitectureElement* and respectively *VA*.

As will be detailed in Section 4.3.5, the matching algorithm uses the naming information. Therefore, a common naming scheme is essential. This imposes additional constraints on the architect's modeling viewpoint. It is thus necessary to make use of approaches, which ensure that the value of the *ArchitectureElement*'s naming attribute conforms to the common naming scheme.

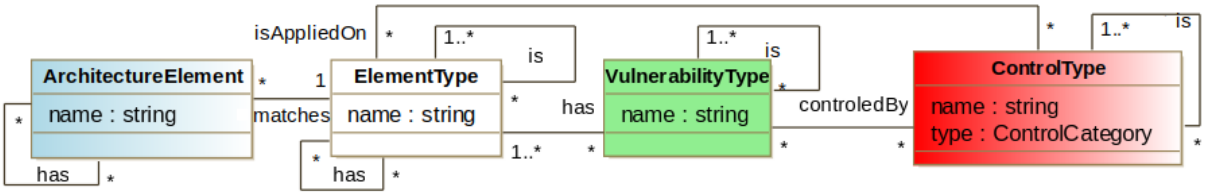


Figure 4.9 – Refinement and Structural Mechanisms

Potential ADLs with which we may consider integrating our assistance approach have thus to provide mechanisms of enforcing and/or verifying this naming scheme. Alternatively, this would be left at the charge of the architect.

4.3.4.4 Refinement and structural mechanisms

The architect defines the architecture iteratively, progressively refining it from a more abstract architecture to a more (partially) concrete one. As part of this architecture, the *name* of the *ArchitectureElement* needs to *match* the *name* of an *ElementType* existing in our knowledge base, as shown in Figure 4.9. As an *ArchitectureElement* becomes more concrete with time, the *ElementType* needs to mirror this evolution. Therefore, there is a need to model an abstraction hierarchy among *ElementTypes*. We model this with the help of the *is* relation.

In our motivating example, the domain specific architecture model contains an *ArchitectureElement* "Aerospike Database Server 3.10.0.3". This *matches* an "Aerospike Database Server" *ElementType*, which *is* a refinement of the "Database Server" *ElementType*. The reason of identifying more abstract level of *ElementType* is because not all vulnerabilities of specific product are revealed in the current security breach repositories (not exploited by any attack yet), but it doesn't mean that the vulnerability doesn't exist. Identifying the type or family of products (*ElementTypes*) may help provide information about the family's *VulnerabilityType*, which gives an idea about possible vulnerabilities for specific product.

Similar considerations about the abstraction levels hold for *VulnerabilityTypes* and *ControlTypes*. Vulnerabilities are usually concrete and correspond to concrete *ArchitectureElements*, i.e. the vulnerabilities from the repository CVE. However, there are more abstract vulnerability types, such as those proposed by repositories like CWE. We model *VulnerabilityType* similarly with the abstraction levels of the *ElementType*, through an *is*

relation. In our motivating example, the concrete *ArchitectureElement* "Medtronic 24950 MyCareLink Patient Monitor" contains a concrete vulnerability "a hard-coded operating system password" (CVE-2018-8870). This concrete vulnerability is modeled as a *VulnerabityType*. It *is* a refinement of a more abstract *VulnerabilityType* "Use of Hard-coded Credentials" (CWE-798), which in turn *is* a refinement of the *VulnerabilityType* "Improper Authentication" (CWE-287).

Moreover, an *ArchitectureElement* may be composed of other *ArchitectureElements*. We model this through the *has* relation. Similarly, *ElementTypes* may also be composed of other *ElementTypes* (*has* relation). Identifying the *VulnerabilityType* of a component can help enrich the vulnerability information about the *ElementType* containing this component.

When considering the integration with an ADL, the modeling of the generalisation (*is*) and composition (*has*) relations may impose constraints on some ADLs. Those ADLs which enforce the definition of their architecture elements types need to conform to the relations between the *ElementTypes*, like how they are defined in our knowledge base.

As our assistance is based on asset, CPE helps to name assets in a standardized way in order to ease name matching task in the assistance database. Moreover, IT products in CPE are referenced with vulnerabilities in CVE if vulnerabilities of these products have been exploited and reported. This reference can ease our assistance knowledge integration.

Table 4.2 synthesize the definitions of concepts used in the security assistance data model.

4.3.5 The security assistance process

The data model presented above unifying concepts from the different viewpoints allows us to integrate the knowledge specific to the attacker and the defender into our security assistance. In this section, we present in detail how the security assistance process can be enacted to assist the architect based on this data model.

The security assistance process is presented using an enhanced BPMN diagram as shown in Figure 4.10. It consists of two major phases: one performed by the architect, and another performed by the security assistance taking the role of security experts. Note that the yellow parts are independent of the domain architecture.

In the architect's annotation phase, the architect *selects* on the architecture model an *ArchitectureElement* (1), on which he/she *annotates* a *selected SecurityProperty* (2) and *Category* (3), obtaining an *AnnotatedDA*. In the security assistance phase, two tasks are

Aspect	Concept	Definition
Domain Architecture	AnnotatedDA	A domain asset (or architecture element) that is annotated with a security property and a category.
	ArchitectureElement	An element that the architect wants to protect. It is represented in the architecture model.
	SecurityProperty	The security objective that the architect wants to protect, such as confidentiality, integrity and availability.
	Category	The category of the AnnotatedDA to be protected, such as data, system behavior, human related asset and physical asset.
Attack	Attack	An attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of an asset.
	AttackPivot	An attack goal or an attack tactic.
	Goal	An aim or purpose of an attack.
	Tactic	A planned way of conducting an attack
	Root	A pair of a security property with a category, which is the final goal of an attack.
	VA	Anything that is valuable to a security expert. It contains vulnerabilities that can be exploited by attacks.
	VulnerabilityType	A vulnerability, a weakness or a design error that may result in an undesirable event, whose exploitation can compromise the involved VA.
Defense	VDA	Anything that has value for domain experts, but also has vulnerabilities that can be exploited by attacks.
	ControlType	Safeguards or countermeasure, designed to protect the security properties of an asset, and met a set of defined security requirements, is a measure against threat.
Refinement and Structural	ElementType	The general types of elements that we can use as components during the architecture modeling.

Table 4.2 – An inventory of definitions of concepts used in the data model

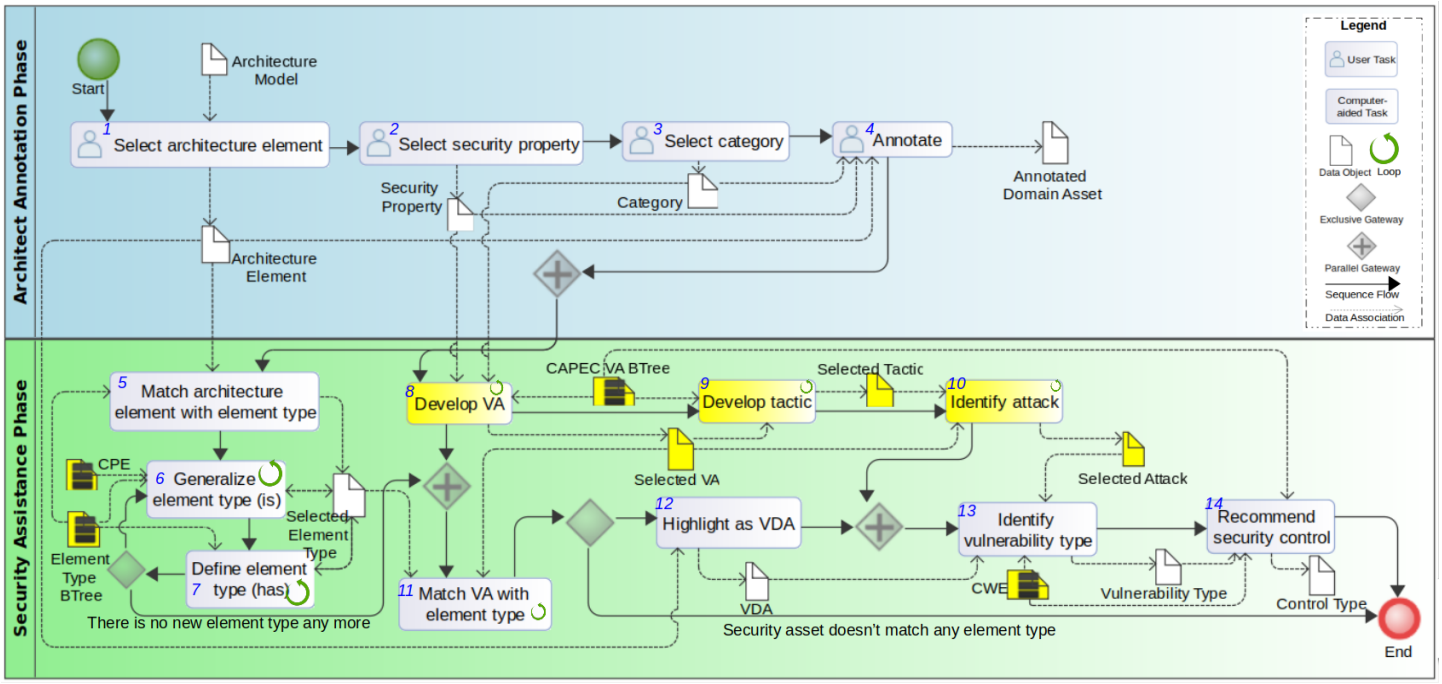


Figure 4.10 – Assistance Process. Each task is uniquely identified with a number placed before its description.

conducted in parallel after the *annotation* (4) task of the architect:

i) The assistance process uses the *ArchitectureElement* to obtain a list of relevant *ElementTypes*. It starts by *matching the architecture element with the element type* (5). The matching algorithm takes as input, on the one hand, the CPE-formated *name* of the *ArchitectureElement*, and on the other hand, the *name* of the *ElementType*. For the correct function of the matching algorithm, a common naming scheme between the *ArchitectureElements* and the *ElementTypes* is necessary. We implement this scheme based on CPE. This matching approach is based on the hypothesis that the architect follows the imposed naming constraints. An alternative solution to allow ADLs make typing of *ArchitectureElements*, would be the use of typing mechanisms. If there is a match, the assistance *generalises* (6) the abstraction hierarchy (described by the *is* relation) into the increasingly more abstract *ElementTypes*. After this, the assistance *defines*, for each of the *ElementTypes* identified in the previous task, of which *ElementTypes* it is composed, according to the *has* relation (7). This repeats until no new *ElementType* is discovered.

ii) In parallel, the assistance uses the APT to *develop (refine and/or decompose) VA* (8). Then it continues to *develop (refine and/or decompose) tactic* (9). The APT begins by the *SecurityProperty* and the *Category* annotated by the architect. These two tasks

result in two lists of *Selected VA* and respectively of *Selected Tactics*. From these two lists, the assistance *identifies attacks* (10).

Based on the list of *Selected ElementType* and on the list of *Selected VA*, our assistance process *matches VA with ElementType* (11). The matching algorithm takes as input, on the one hand, the CPE-formated *name* of the *ElementType*, and on the other hand, the *name* of the *VA*. If the *name* of the *VA* is found syntactically and/or semantically matching with an item in the *Selected ElementType* list, then the assistance *highlights* the corresponding *ArchitectureElement* as a *VDA* (12). The *ElementType* that is related with the *ArchitectureElement* from the architecture model is as valuable as the *Annotated DA* for the architect.

For the enrichment of Element Type List, architects with basic security skills are usually able to be aware of the type of an element and correctly identify its types and its components.

Meanwhile, this *ElementType* is as vulnerable as the *VA* to an attacker and needs protection from the security defense expert as the *VDA*. This triple nature of the *ElementType* makes it the bridge among the architect, the attacker and the defender's viewpoints.

Using the identified *VDA* and a vulnerability repository (e.g. CWE) as input, the assistance *identifies VulnerabilityTypes* (13). Once the list containing the concerned vulnerability types is enriched, the assistance *recommends security ControlTypes* corresponding to these *VulnerabilityTypes* to the architect (14) according to mitigation information from CAPEC and CWE.

This process can also be presented in the form of algorithm with pseudo code. Algorithm 1 shows the security assistance process algorithm.

In Algorithm 1, the assistance tool receives an architecture element *ae*, a security property *sp* and a category *c* from a architect. Meanwhile, the assistance database contains:

1. a list of general element types (ET);
2. a 2-dimensional array vulnerable asset (VA) where all vulnerable assets (va) are children of a *root*, which is the concatenation of a c and a sp (line 17);
3. a map table CC between *container* and *component*, both of them belong to ET;
4. a map table VV between va and vulnerability type (vt);
5. a map table VC between vt and control (ct).

The algorithm outputs a list of vulnerable domain asset (VDA), a list of vulnerability type (VT) and a list of control (CT) to propose security recommendations to the architect.

Algorithm 1 Assistance Process

Input

ae, sp, c; /* architecture element, security property, category*/
array ET of n strings; /*list of element types in assistance database*/
2-dimensional array VA[root, va]; /*list of vulnerable assets under different roots in assistance database*/
map CC[container, component]; /*a map of element types with has-a relation*/
map VV[va, vt]; map VC[vt, ct];

Output

VDA[]:vulnerable domain asset list; VT[]:vulnerability type list; CT[]:control list;

```
1: Declare arrayList LS
2: for i=0 to n -1 do
3:    if ae=ET[i] then
4:       LS.add(ET[i]);
5:    end if
6: end for
7: for all element type j in LS do
8:    if LS[j].parent != NULL then
9:       LS.add(LS[j].parent);
10:   end if
11: end for
12: for all element type j in LS do
13:   if LS[j] in CC.container then
14:      LS.add(CC.component);
15:   end if
16: end for
17: VA.root← c+sp;
18: for all element type j in LS do
19:   if LS[j] in VA.va then
20:      VDA.add(LS[j]);
21:   end if
22: end for
23: for all vda in VV do
24:   VT.add(VV.vulnerabilityType);
25: end for
26: for all vt in VC do
27:   CT.add(VC.ct);
28: end for
```

At the beginning of algorithm, we declare a empty list *LS* to store later involved element types (*et*) (line 1). If *ae* is found in *ET*, then we add its corresponding *et* into *LS* (line 2-6). Next, for each *et* in *LS*, if the parent of *et* is not null, which means that we can generalize *et* in a more abstract *et* (is-a relation), then we add this parent into *ES* (line 7-11). For each *et* in *LS*, we search if an *et* is found in *container* column of table *CC*, if it is the case, which means it contains *component*, which is also an *et* (has-a relation), then we also add the *component* into *LS* (line 12-16). Now a list of *et*s that are involved by the *ae* is ready for further analysis. Thereafter, the *root* of "attack pivot tree" (*apt*) is determined by the concatenation of *c* and *sp* (line 17). For each *et* in *LS*, if the *et* is found in *va* of *VA*, which means the *et* matches a vulnerable asset that is a potential target of an attack, then we add this *et* into *VDA* (line 18-22). For each vulnerable domain asset (*vda*) in *VV*, we search its corresponding *vt* in *VV* and add it into *VT* (line 23-25). For each *vt* in *VC*, we search the corresponding *ct*s and add them into *CT* (line 26-28).

It is worth noting that given two graph-based structures (Selected Element Types and APT in our case), a matching activity is to produce a mapping between the nodes of the two structures that syntactically or semantically correspond to each other [GSY04]. As we can see, there are two types of matching : syntactic matching and semantic matching. In syntactic matching, the relations between nodes of the two structures are computed between label of nodes; while in semantic matching, the relations are computed between concepts of nodes.

The natural language is ambiguous to precisely present the nodes' semantics. The natural language is considered as an external language. Contrary to external language, internal language, which has defined syntax and semantics, is used to express concepts [GSY04]. Therefore, to allow matching between "Element Types" and "Vulnerable Asset", semantic matching can be utilized, which often solves semantic heterogeneity problem, namely managing the diversity in knowledge. Collaboration among people of different geographies and languages, having different viewpoints and using different terminology has always been a huge problem. This problem is aggravated with the advent of the Web and the consequential information explosion. That's why semantic matching is privileged than syntactic matching.

The tasks in Figure 4.10 can be almost completely enacted thanks to the data model introduced in the previous section. In fact, as suggested in Figure 4.10 and outlined in the following description, only the tasks in the architect's annotation phase and the matching activity require some limited user intervention, while the others can be carried out by

only leveraging the information provided by the assistance knowledge base.

More precisely, the security assistance relies on the availability of a complex knowledge base named VA library, which will be introduced in Section 5.3. This VA library collects vulnerability, threat and security control information for associated vulnerable assets. Likewise, the VA library maps the vulnerable assets to vulnerability types and to a list of security controls that can be used as countermeasures to mitigate the threats that exploit the vulnerability types. This allows to retrieve the full list of security controls associated with the identified vulnerable assets. It is worth outlining that the VA library is continually enriched by security experts, new best practices and new standards.

4.4 Conclusion

In this chapter, we have introduced the first contribution of this thesis, which tries to bridge the knowledge gap between architects and security experts, from the architect's viewpoint, by proposing a security assistance. We have illustrated our security assistance by using a telemonitoring system motivation example. To enact this security assistance, we first describe that this bridge is based on a novel refinement of the concept "asset". Then, according to this philosophy, we have introduced an asset-based security assistance framework to show the general view of this assistance, together with a data model to structure relevant concepts and their relations, and a BPMN-based process to help enact the security assistance. This security assistance can be used by architects, who have limited security knowledge, to integrate the security aspect into the architecture modeling. In the next chapter, we present our second contribution of this thesis, which aims to bridge the gap from the security expert's viewpoint, by structuring the widely-used threat modeling process, especially the asset identification phase in the threat modeling process.

Chapter 5

Contribution 2: Security expert's viewpoint: A structured threat modeling to bridge the knowledge gap between security experts and architects

Contents

5.1 Introduction

In this chapter, we address the second contribution of this thesis, aiming at bridging the gap between architects and security expertsn, from the security expert's viewpoint. For that, we intend to structure and improve the threat modeling process, which is widely-used to integrate the security into SDLC. In this thesis, we aim at improving the threat

modeling process by structuring the asset identification phase, with a focus on the design stage in SDLC.

Threat modeling is recognized as one of the most important activities in software security. It helps to address security issues in software development. Threat modeling processes are widely used in the industry such as the one of Microsoft SDL. In threat modeling, it is essential to firstly identify assets before enumerating threats, in order to diagnose the threat targets and spot the protection mechanisms. Asset identification and threat enumeration are collaborative activities involving many actors such as security experts and software architects. These activities are traditionally carried out in brainstorming sessions. Due to the lack of guidance, the lack of a sufficiently formalized process, the high dependence on actors' knowledge, and the variety of actors' background, these actors often have difficulties collaborating with each other during these sessions.

Brainstorming sessions are thus often conducted sub-optimally and require significant effort. To address this problem, we aim at structuring the asset identification phase in this chapter by proposing a systematic asset identification process, which is based on a reference model that structure the relevant concepts and their relations. This asset identification process structures and identifies relevant assets, facilitating the threat enumeration during the brainstorming session. We present this structuring of threat modeling process in the first part of this chapter.

To identify relevant assets, which can be domain, vulnerable and vulnerable domain assets, it is necessary to build a vulnerable asset library to record common vulnerable assets together with their vulnerability types, threats and security controls. This vulnerable asset library should be able to be reused to link with domain assets from different domains. In the second part of this chapter, we illustrate a structured process of extracting vulnerable assets from security repositories such as CAPEC to build this vulnerable asset library.

5.2 Structuring the asset identification phase in the threat modeling process

The state of practice in threat modeling has largely remained ad-hoc, driven by manual labor, relying to a large extent on the level of expertise of the participants involved [Ysk+20; Joh+16]. Consequently, the threat modeling process and its results are subjective and are hardly reproducible.

Our proposal addresses the 3 threat modeling challenges and/or requirements identified in Section 3.2.2.3. For this, we aim at structuring the asset identification phase: to associate it with a well-defined process, promoting the manipulation of a set of precise and well-structured concepts and exploiting, if needed, a security knowledge base to limit the negative impact of the lack of experience in security. These elements can be used to guide the participants during the brainstorming sessions.

This proposal is based on the novel refinement of the concept of *asset* which we have described in Section 4.3.1. We show how we use this novel refinement to structure the universe of threat modeling knowledge by designing a reference model. This reference model is used to structure the common language of the information handled by all participants during this phase. Besides, it can also be used as a language with which one can capitalize in a knowledge base the state of the art in security. The concepts in the reference model can be customized and instantiated in different architectural contexts to define specific assets and threats to a system. Based on this reference model, we next present an asset identification process to lead the asset identification phase and guide the participants.

5.2.1 Structuring the domain and security knowledge – An asset-based reference model

Now that we have refined the *asset* concept, we propose to structure the universe of information manipulated during the brainstorming sessions in threat modeling, using a *reference model*. The definition of this reference model has two objectives:

— Fixing and structuring the discourse during brainstorming sessions;

— Allowing the capturing of security knowledge from the literature in a form which can be then reused during brainstorming.

This reference model is presented in Figure 5.1. *Asset* is the core concept of this model. It is an abstract class. As we have discussed above, we specialize the concept of *Asset* into *DomainAsset* (DA) and *VulnerableAsset* (VA). The *VulnerableDomainAsset* concept (VDA) is a type of both *DomainAsset* and *VulnerableAsset*. Both *VulnerableAssets* and *VDA* can *have Vulnerabilities*, which can be *exploited* by *Threats*. Thus *Threats* can *compromise* VA and thus VDA. In its compromise actions, a *Threat* may *target* an *Asset* (both *Domain* and *Vulnerable*) *using* other compromised *Assets* in the process. To mitigate the *Vulnerabilities*, *Controls* can be *applied* on *VDA*.

In Figure 5.1, each *Asset* can have three relationships with other Assets: *is*, *has* and

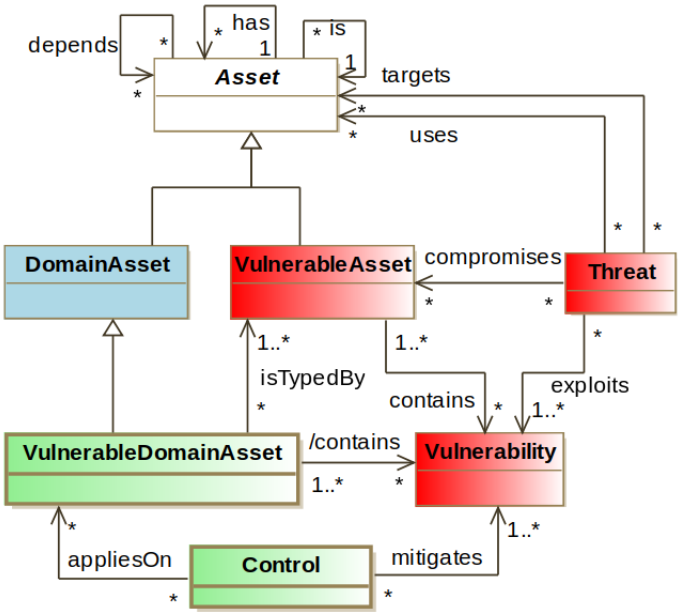


Figure 5.1 – Asset-based reference model

depends. The *is* relation captures the generalisation between *Assets* of different abstraction levels. It captures an iterative refinement of assets. For example, the domain experts or the architects define the list of domain assets coming from the domain architecture model. During the design phase, architects can progressively refine this list from more abstract assets to more concrete ones. Similarly, the security experts define an hierarchy from more abstract (coming from abstract attack pattern) to more concrete vulnerable assets. Moreover, an *Asset* may be composed of other *Assets*. We model this through the *has* relation. We also introduce the *dependency* relation between *Assets*. A dependency exists between two *Assets* if changes to one *Asset* (the supplier) may cause changes to the other (the client) [Fow03].

It is worth noting that dependencies between *Assets* can link two different security properties in two different assets. For example the integrity of the server software can directly affect the availability of the data traited by that server, or affect the confidentiality of a user's information [FS09]. Therefore, this is another reason why analyzing asset can be an easy way to connect security requirement and understand attacker's perspectives.

These three relationships (generalisation, composition and dependency) are very common in system architecture modeling. It is worth noting that this kind of modeling promotes a data structure similar to that of a B-tree [BM70], even if other data structures are

also possible, such as class diagram. We choose B-Tree structure because it can be easily coupled with and extended from Attack Pivot Tree, as we deal with security aspect. A B-tree is a tree data structure where each level can have one or more *children* nodes. Each node may be thought of as a kind of list, containing several entities called *keys* (related to the origin of B-trees for databases).

In our case, the *Assets* related by an *is*, are similar to the *children* nodes of a B-tree. For example, in Figure 5.2, which shows a short structure of A vulnerable asset (VA) B-Tree, VA2 and VA6 are the *children* of VA1, and VA3 is a *child* of VA2. The *Assets* related by *has* and *depends*, correspond to the *keys* of a B-tree. VA4 and VA5 are *keys* of VA2, related respectively by *has* and *depends* relations. In our data structure, we just take inspiration from the idea of B-trees, but are not interested in their properties, such as self-balancing.

We consider B-Tree structure (e.g. VA B-Tree) is an improvement of the APT presented in Section 4.3.3, because it allows to show all the three relations of different dimensions inside the same tree structure. More precisely, B-Tree allows showing the generalization relation vertically, and showing composition and dependence relations horizontally inside each child node, to align with the relations of different dimensions among different assets. Moreover, this is close to the structure of existing security knowledge repositories, such as CAPEC, facilitating the structuring of vulnerable assets extracted from them (cf. Section 5.3).

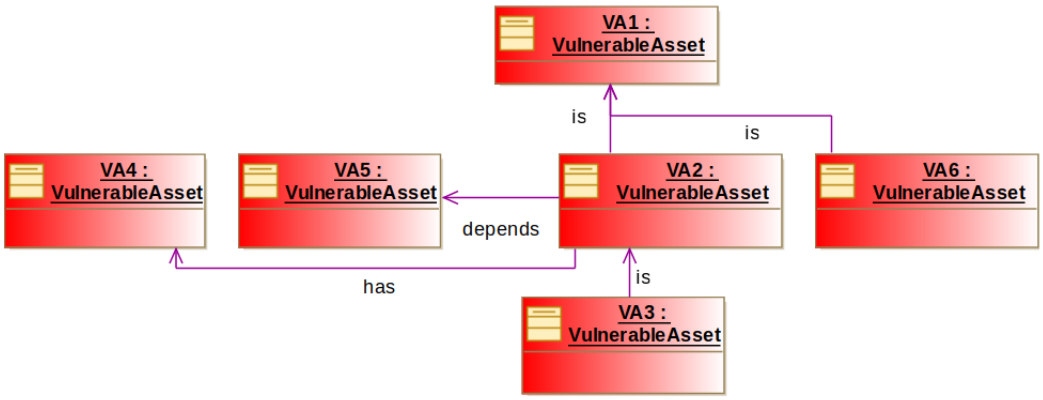


Figure 5.2 – Vulnerable Asset (VA) B-tree

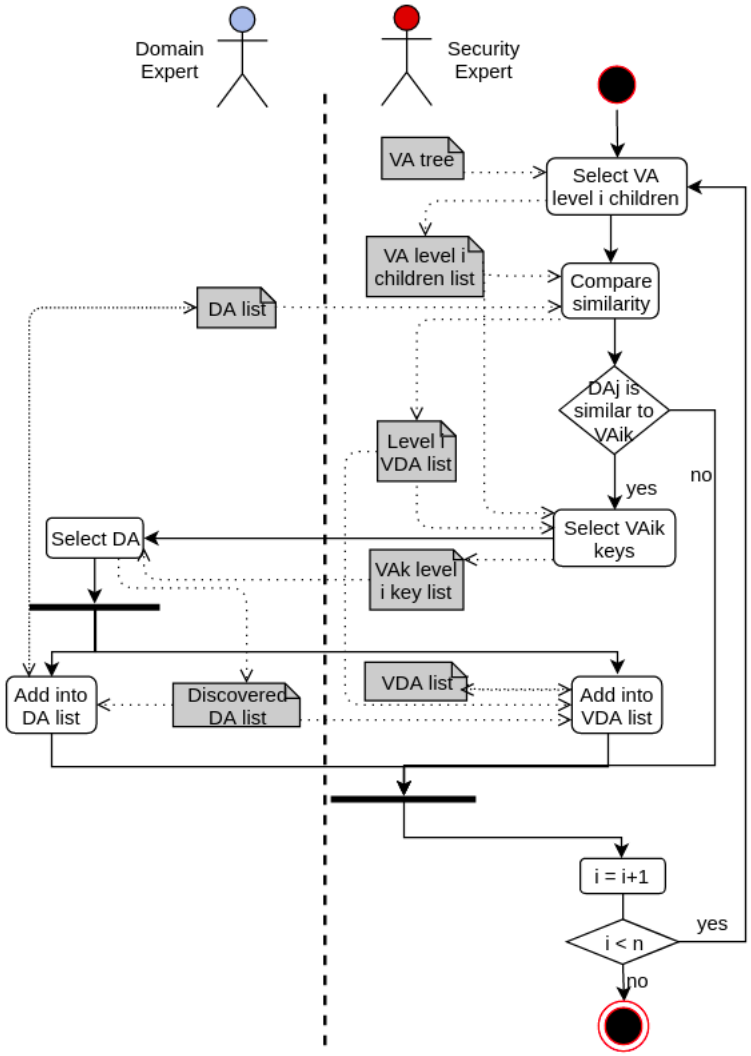


Figure 5.3 – Asset identification process

5.2.2 Defining an asset identification process

After refining the *Asset* concept and proposing an asset-based reference model to structure the domain and security knowledge, we introduce an asset identification process to help threat modeling participants in the identification of assets and threats targeting these assets. This process is shown in Figure 5.3 and the general view behind this idea has been summarized in Figure 4.2. This process can be launched regardless of the software development stage and therefore on more or less abstract models.

On one hand, Domain Assets (DA), obtained from domain architecture models such as enterprise, system and software architectures, is structured by relationships such as

generalization, composition and dependency. On the other hand, security experts identify Vulnerable Assets (VA) that are relevant to the types of elements present in the model being designed. This list of VA can be populated from the security experts' knowledge, as well as be extracted from common security knowledge repositories, thus promoting reuse. This extraction is a non-trivial process, involving threat libraries, attack patterns (e.g. CAPEC), attack trees, vulnerabilities, etc. Thus, we promote setting up a VA library synthesizing current knowledge of the field in a format that respects our reference model. More details about this VA library are presented in Section 5.3.

In our asset identification process, both the domain and the vulnerable assets are structured similarly to a B-tree. However, it is not necessary to consider this for the domain assets, but rather treat them, for simplicity reasons, as a list. On the other hand, we use this B-Tree structure for the vulnerable assets.

Figure 5.3 illustrates a fine-grained asset identification process. It takes as inputs a DA list and a VA B-Tree. The DA list is a result of domain experts and/or architects identifying assets specific to their domain architecture which are valuable. The VA B-Tree results from the security experts using their knowledge to identify generic vulnerable assets, and it can be enriched with information extracted from security knowledge bases or a VA library. The goal of this asset identification process is to identify VDA, together with corresponding their threats and vulnerabilities, and to propose security controls.

The asset identification process can traverse the vulnerable asset tree (respecting to B-tree) either in a depth-first or in a breadth-first manner. In this thesis, we choose to present it with the breadth-first strategy:

— As such, the process *selects* the VA B-Tree children situated at the current vertical "i" level (i.e. all the VA linked through an *is* relation to the VA of the previous parent level). For instance, when considering the example in Figure 5.2, concerning the level "i=1", the children are VA2 and VA6;

— For each VA child, the domain and security experts *compare* its syntactic and semantic similarity with each DA in the current domain asset list. If a VA_k, from the list of VA level i children, is found similar with a DA_j, from the DA list, further similarities are searched;

— To search the further similarities, the VA B-Tree is traversed horizontally, and the keys attached to that VA_k are *selected* (i.e. the VA linked through *has* and *depends* relations). Let us suppose that for the example in Figure 5.2, if VA2 is found similar with a DA, then its key list containing VA4 and VA5 is *selected*;

— The domain experts select among VA_k keys those which are involved in the domain. VA_k keys that are involved in the domain, discovered at this "i" iteration, are added to the current DA list, enriching the DA list for the next iteration;

— As they are initially VA, but also in the domain, they are actually VDA, and therefore are also added to the VDA list. Let us suppose that in our example VA4 is a key that is involved in the domain. At the end of the "i" iteration, the DA list additionally contains VA4 and the VDA list contains VA2 and VA4;

— Then, if no more VA_k key is found being involved in the domain, the process advances to the next VA tree level (i.e. "i=i+1"), until there are no more levels (i.e. "i==n"). The DA and VDA list are enriched with the iteration of each "i" level.

Since DA and VA are similarly structured by the *is*, *has* and *depends* relationships defined in the reference model, the goal of the asset identification process is to bridge the gap between these two sets of assets by identify the mapping between DA and VA (cf. Figure 4.2) and identify VDA (i.e. Domain Assets which are also Vulnerable). For that, the security experts project or instantiate these VA on the DA. Then a comparison is made by actors to identify if a mapping occurs between VA and DA. If mappings are identified, they represent a VDA. It is therefore noteworthy that the matching activity instantiates *abstract* VA into *concrete* VDA.

Figure 5.4 shows an excerpt of the matching activity between the nodes from Element Type B-Tree and the nodes from VA B-Tree. In this example, Element Types are collected from the IoT domain (an example domain), while Vulnerable Asset from VA B-Tree are extracted from CAPEC. As we can see, we have identified three matchings, thus three VDA, they are respectively "Interceptable Intent in Android's Broadcast Receiver", "Modifiable Intent in Android's Broadcast Receiver", and "Modifiable GPS Signal in Raspberry Pi's GPS Module".

Once the VDA list has been enriched, it is used as a bridge towards the next threat enumeration phase in the threat modeling process. Security and domain experts may use it to identify security mitigations to the identified vulnerabilities. To further help the participants, threat libraries containing vulnerabilities of assets, threats exploiting these vulnerabilities and controls mitigating the vulnerabilities can be used. In this way, our approach uses domain-independent, general, security threat and attack knowledge to identify and protect domain-specific VDA. The threat libraries including VA library can be populated by extracting information from security knowledge repositories using approaches such as the one presented in the next subsection.

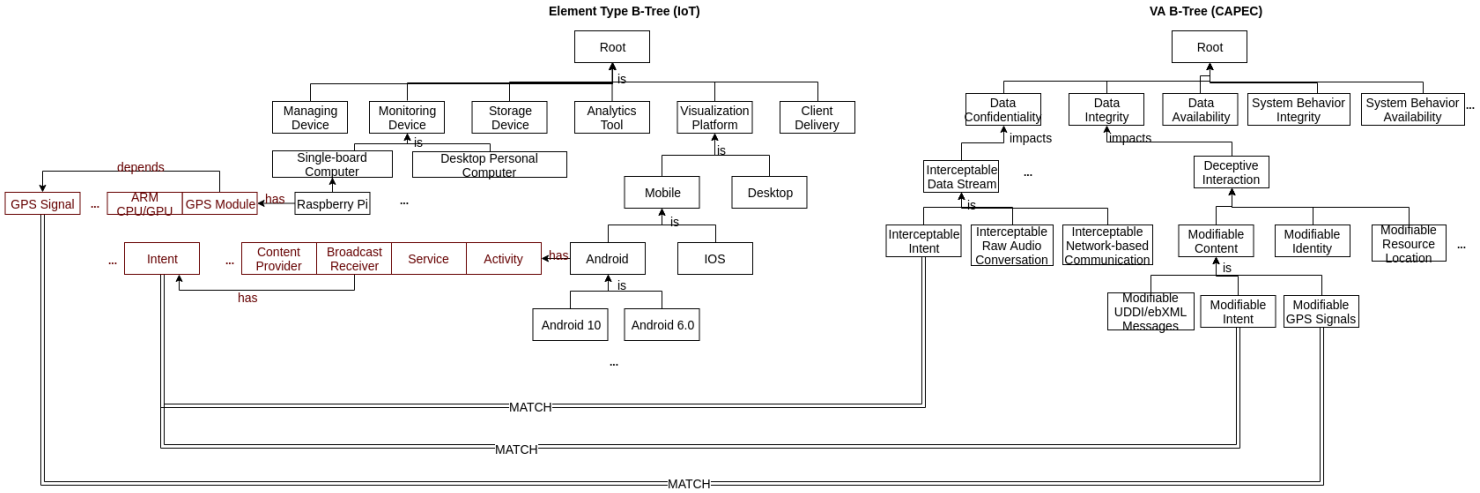


Figure 5.4 – Element Type B-Tree matches VA B-Tree

5.3 Building a vulnerable asset library following a structured process of extracting information from the security repository CAPEC

To reduce the level of security expertise required, threat modeling can be supported by threat libraries (structured or unstructured lists of threats), which have been found particularly effective in industry scenarios [UF14]; or by attack taxonomies, which offer a classification scheme to help actors find relevant attacks more easily.

However, non-security experts, such as domain experts (including architects), have to be trained to better use threat libraries, as they require a minimum security knowledge to understand the security jargon. Therefore, we believe that it is useful to construct a vulnerable asset library, which can enrich the threat library, and can be coupled with domain assets and thus can help the asset identification process, in order to lessen the effort of non-security experts.

The VA library aims to classify a wide variety of abstract, system- and technique-independent vulnerable assets, which keeps the asset identification and threat enumeration manageable, increases the VA library's applicability and reusability, and makes it both more practical and more useful for security novices and experts alike. For the library to be well integrated with the asset identification process, we propose that the library and the reference model presented in Section 5.2.1 follow the same structure for the vulnerable

assets.

As we have mentioned above, Vulnerable Asset (VA) represents security experts' viewpoint and it is domain-independent. Therefore, VA can be extracted from existing security knowledge repositories for the reuse in different contexts or domains. In this section, we stress the importance and reusability of this extraction and present part of extraction rules by leveraging well-known attack pattern knowledge bases such as CAPEC and by respecting the B-tree structure. This extraction process is considered as a prerequisite for the successful asset identification process presented in Section 5.2.2.

Attacks are possible realisations of threats [SB14]. Therefore attack descriptions can be useful in enumerating threats. To construct this threat library, we can leverage existing attack repositories such as CAPEC [MITb], OWASP [Fou17] and ATT&CK [MITa], as these repositories describe attack patterns. However, they all describe attack patterns in natural language, in an unstructured way and possibly ambiguous, which make them difficult to be processed automatically. At the current state of the advancement, we identify a number of heuristic rules, which can be enriched in the future. These rules help partially extract vulnerable assets that can be compromised by threats, and their relations.

We show these extraction rules by leveraging CAPEC, which is one of the most popular and structured attack repositories. In CAPEC, the attacks belong to different levels of abstraction: *view, category, meta, standard and detailed*. As *view* and *category* levels are too abstract to be reused effectively, we focus on the *meta, standard and detailed* abstraction levels:

— A *meta* attack pattern is "an abstract characterization of a specific methodology or technique used in an attack", and "a generalization of related group of *standard* attack patterns";

— A *standard* attack pattern is "focused on a specific methodology or technique used in an attack";

— A *detailed* attack pattern "provides a low level of detail, typically leveraging a specific technique and targeting a specific technology, and expresses a complete execution flow".

Detailed attack patterns are more specific than *meta* and *standard* attack patterns.

— The links between these abstraction levels are modeled through "*childOf/parentOf*" relations. This hierarchical attack/threat structure can help us identify *is* and *has* relations between vulnerable assets;

— Moreover, there are also relations of "*canFollow*/*canPrecede*" between attacks/threats in CAPEC, which can help us identify *depends* relation between vulnerable assets.

Based on CAPEC attack natural language descriptions, we define several VA extraction rules:

— **Rule 1**: If the name of attack pattern contains the keyword "contaminate", or "poison", or "leverage", or "manipulate", or "abuse", or "exploit" or "misuse", etc., then the noun set after any of these keywords is selected as a vulnerable asset (VA). For example, for the *detailed* attack pattern "Poison web service registry" (CAPEC-51), the "web service registry" is a vulnerable asset;

— **Rule 2**: If the name of attack pattern contains the keyword "manipulation", or "poisoning", or "tampering" or "alteration", etc., then the noun set before any of these keywords is extracted as a vulnerable asset (VA). For example, for the *standard* attack pattern "Web service protocol manipulation" (CAPEC-278), the "web service protocol" is a vulnerable asset;

— **Rule 3**: If the name of attack pattern contains the keyword "injection", or "inclusion" or "insertion", etc., then the noun set before any of these keywords is selected and we add the literal "Untested" before and "Input" after this noun set, the whole literal word is considered as a VA. For example, for the *standard* attack pattern "XML injection" (CAPEC-250), "XML" is selected and added by the above prefix and suffix. As a result, "UntestedXMLInput" is a vulnerable asset.

The VA rules are generated with a string manipulation approach. The list of the above keywords can be enriched. The whole list of keywords that we have identified from CAPEC can be found here [1].

As mentioned in Section 5.2.1, there are three possible relations (*is, has, depends*) between vulnerable assets. By leveraging CAPEC, we can also extract the relations between vulnerable assets.

— **Rule 4**: The "childOf" relation between two attack patterns is translated into either "*is*" or "*has*" relation between two corresponding vulnerable assets, because "ChildOf" in CAPEC can present either a specialisation or a decomposition relation. For example, on one hand, the "SOAP" vulnerable asset extracted from the *detailed* attack pattern "SOAP Manipulation" (CAPEC ID 279), *is* a type of "Web Services protocol" vulnerable asset. On the other hand, the "XML" vulnerable asset

1. https://github.com/lunanan/ArchwareExtraction/blob/master/ArchwareExtraction/keywords.txt

has "DTD", "XPath" and "XQuery" vulnerable assets, extracted respectively from three *detailed* attacks (CAPEC IDs respectively 228, 83, 84). Therefore, the reasoning about the decision comes from the security experts who extract vulnerable assets;

— **Rule 5**: The "canFollow" relation between two attack patterns is translated into *depends* relation between two relevant vulnerable assets, because if asset A_a is compromised by an attack/threat T_a, then a threat T_b, which can follow T_a, can compromise asset A_b, therefore asset A_b *depends* on asset A_a.

These rules allow us to extract vulnerable assets from attack/threat patterns. For each extraction, the relation between the threat and the VA is stored. This allows to later find all the threats that compromise the same VA. In this way, our library contains the information about VA and threats that compromise them. At the current state of this thesis, we have implemented the above rules in a BASH scripting application for the VA extraction process [2]. We believe that this extraction process can be improved using other techniques such as parsing and text mining to allow automation, which is our future work.

The VA library, as a part of threat library, lightens the dependency on security knowledge. It aims to be utilized by both security and non-security experts. Therefore, the construction of the VA library can satisfy to the requirement 2 in Section 3.2.2.3. The VA library can thus help identify threats relevant for the domain elements, through the intermediary of identified VDA.

5.4 Conclusion

In this chapter, we aim at bridging the gap between architects and security experts from the security expert's viewpoint, by structuring the asset identification phase in the threat modeling process. Threat modeling is a result of a collaborative process involving many actors from different backgrounds. Despite its importance, bridging the knowledge gap between architects and security experts during their collaboration in threat modeling is not trivial. One of the main reasons is that threat identification and enumeration is often a challenging task for non-security experts. Thus, architects have to rely on threat modeling processes, which may quickly turn into a complex task when these processes lack guidance and formalisation.

2. https://github.com/lunanan/ArchwareExtraction

To address this limitation, we have proposed a reference model and a systematic asset identification process to facilitate the collaboration between actors from different backgrounds, to guide participants and to structure the asset identification phase. As a result, pertinent assets such as Vulnerable Assets are structured and Vulnerable Domain Assets are identified to improve the threat enumeration phase. Then, we have discussed how we can build a vulnerable asset library following a structured process of extracting information from the security repositories such as CAPEC. This VA library aims to classify a wide variety of abstract, system- and technique-independent vulnerable assets, which keeps the asset identification and threat enumeration manageable, increases the VA library's applicability and reusability, and makes it both more practical and more useful for security novices.

Part III

Proof-of-Concept

Chapter 6

Security Assistance Proof-of-Concept

Contents

6.1 Introduction

In this chapter, we evaluate our first contribution with three steps:

1. Illustrate the feasibility of the security assistance's enactment by applying it on the motivating example and showing the results;
2. Secondly, we show an assistance prototype web application that we have implemented dedicated to help architects;
3. Finally, we conduct a crossover experimentation with master students in software engineering field, playing the role of architects, we choose the crossover study because crossover designs are widespread in software engineering experimentation [VAJ15]. They require fewer subjects and control the variability among subjects.

6.2 Assistance enactment: Application of the security assistance on the motivating example: the results

To provide concrete application examples of the proposed security assistance and to show its enactment and its usefulness, we apply it on three annotated DA in the motivating example to illustrate that this security assistance is able to deal with both concrete and abstract aspects. We firstly show how we have developed a database enabling the query-based simulation of the assistance process. The results of this simulation for each of the three annotated DAs of the motivating example are presented in the next sections.

As the data model in Figure 4.5 used by our assistance process integrates knowledge from several databases, we chose to implement it using a database. The tasks of the process described previously in Figure 4.10 are implemented as "SQL select statements". Our assistance database is filled with information extracted from several existing, widely-used repositories. As such, the *VAs* and *Relations* are extracted from CAPEC. Some of VAs are extracted according to the rules proposed in Section 5.3, others are extracted manually by a security expert. The *VulnerabilityTypes* are extracted from CWE, and the *SecurityControls* are extracted from both CWE and CAPEC. The names of the *ElementTypes* are extracted from CPE. The extraction process is, for the moment, mainly based on security experts' knowledge. At the current state of this thesis, our database contains a part of the knowledge of the existing repositories. For example, our assistance database contains the

knowledge associated with 148 attack patterns from the 533 attack patterns that CAPEC contains, and with 88 vulnerabilities out of the 1141 that CWE contains.

6.2.1 Assistance on *DA1*: concrete security control recommendation on a concrete architecture element

The results of applying the security assistance process on the three *Annotated DAs* in the motivating example are presented in Table 6.1. Each row corresponds to each *Annotated DA*. Each column corresponds to a data object obtained from a task of the assistance process. The number of the corresponding task is indicated in brackets. In the first case of our motivating example, we consider three parts (same for the other two cases):

— The DA parts:

  — The architect *selects* the *ArchitectureElement* "Medtronic 24950 MyCareLink Patient Monitor" (the task *1 Select architecture element* in Figure 4.10);

  — The architect also *selects* (2) the *SecurityProperty* "Integrity" and *selects* (3) the *Category* "Data", with which he/she *annotates* on the "Medtronic monitor", obtaining the *AnnotatedDA* (4) {Medtronic 24950 MyCareLink Monitor, Integrity, Data};

  — Taking as input the *ArchitectureElement* "Medtronic 24950 MyCareLink Patient Monitor", our assistance process obtains a list of *ElementTypes*, starting with an initial *match* (5) with the current list of *ElementTypes* in our database. This initial match is in this case an *ElementType* named "Medtronic 24950 MyCareLink Patient Monitor";

  — The *ElementType List* is enriched following *generalisation* (is) (6) and *composition* (has) (7) relations, containing numerous items, among which we cite here only four: the "Medtronic monitor" itself, "Monitor" as a generalisation of "Medtronic monitor", "Mobile" as a generalisation of the "Monitor" and "Hard-coded password" as a constituent of the "Monitor".

— The VA parts:

  — In parallel, starting from the *SecurityProperty* "Integrity" and the *Category* "Data", considered as the *Root* in APT (c.f. Section 4.3.3), our assistance

expands/develops (8) it into a tree of *VAs* containing, among other: "Information", "Configuration detail", "Software Structure and Composition", "Compiled object", "Executable", "Machine instructions", "Hard-coded credential" and "Hard-coded password", following the hierarchy of CAPEC;

- For the most concrete *VA*, which is "Hard-coded password", the APT is further *developed with Tactics* (9) that compromise it, such as: "Reverse engineering", "White box reverse engineering" and "Read sensitive strings within an executable";
- From this *Tactics* list and from the *VA* list obtained from (9 and 8), the assistance *identifies attacks* (10), such as "Hard-coded password realized by reading sensitive strings within an executable".

— The VDA parts:

- By *matching* (11) the list of *VAs* with that of *ElementTypes*, the assistance obtains the "Medtronic Monitor Hard-coded password" as a common item. It therefore *highlights* it (12) as a *VDA*;
- For this *VDA* and the identified list of *Tactics*, the assistance *identifies VulnerabilityTypes* (13), among which "Use of hard-coded credentials" (CWE-798);
- Based on this list of *VulnerabilityTypes*, the assistance may *recommend* (14) several *ControlTypes*, extracted from CAPEC and CWE. For example, for the *VulnerabilityType* "Use of hard-coded credentials" (CWE-798), the *ControlType* list contains among other: "Utilize a first login mode" and "Store credentials outside of the code in a well protected encrypted configuration database".

To sum up, the security assistance highlights at least the "Medtronic Monitor Hard-coded Password" as a VDA, a concrete constituent of the concrete architecture element "Medtronic monitor". It also alerts the architect at least the existence of *VulnerabilityType* "Use of hard-coded credentials", and recommends at least two concrete *ControlTypes* "Utilize a first login mode" and "Store credentials outside of the code in a strongly protected encrypted configuration file or database".

	(4) Annotated DA			Element Type List		(8) VA	(9) Tactic	(10) Attack	(11,12) VDA	(13) Vulnerability Type	(14) Control Type
ID	(1) Architecture Element	(2) Security Property	(3) Category	(5,6) is	(7) has						
DA1	Medtronic 24950 MyCareLink Patient Monitor (Concrete)	Integrity	Data	-Medtronic 24950 MyCareLink Patient Monitor -Monitor -Mobile	Hard-coded Password ...	-Information -Configuration Detail -Software Structure and Composition -Compiled Object -Executable -Machine Instructions -Hard-coded Credential -Hard-coded Password ...	-Reverse engineering -White box reverse engineering -Read sensitive strings within an executable ...	-Hard-coded password realized by reading sensitive strings within an executable ...	-Medtronic Monitor Hard-coded Password ...	-Use of hard-coded credentials (CWE-798) ...	-Utilize a first login mode -Store credentials outside of the code in a strongly protected encrypted configuration file or database... (Concrete)
DA2	Aerospike Database Server 3.10.0.3 (Concrete)	Confidentiality	Data	-Aerospike database server 3.10.0.3 -Aerospike Database Server -Database Server -Server	-SQL Statement ...	-Data Input Interpretation -Command Input Interpretation -SQL Statement ...	-Blind SQL statement -Command line execution through SQL injection ...	-SQL statement compromised by command line execution through SQL injection ...	-Aerospike Database SQL Statement	-Improper input validation (CWE-20) ...	-Use an "accept known good" input validation strategy... (Abstract)
DA3	Remote Desktop (Abstract)	Availability	System Behavior	-Remote desktop -Desktop	-Web Browser -XML Parser...	-Application functionality -Appropriate memory allocation -XML parser...	-XML entity expansion -XML quadratic expansion...	-XML parser compromised by XML entity expansion...	-Remote Desktop XML Parser	-missing XML validation (CWE-112) ...	-always validate XML input against a known XML Schema or DTD... (abstract)

Table 6.1 – Assistance results for the three cases of the motivating example. Each row corresponds to one of the *Annotated DA*. Each column corresponds to a data object obtained from a task of the assistance process. The number of the corresponding task is indicated in brackets before the name of the data object.

6.2.2 Assistance on *DA2* and *DA3*: abstract security controls on a concrete and respectively abstract architecture elements

The application of the security assistance process on **DA2** and **DA3** is very similar to the application on **DA1**. The only difference is related to the abstraction level of the *ArchitectureElements* and *ControlTypes*. As such, for **DA2**, while the security assistance process is applied on a concrete *ArchitectureElement*, like in the case of **DA1**, it finds abstract *ControlTypes*, such as "Use an accept known good input validation strategy". If the *ArchitectureElement* is abstract, the *ControlTypes* that are proposed can only be abstract as well. This is the case for **DA3**, in which the abstract *ControlType* "Always validate XML input against a known XML schema or DTD" is proposed for the abstract *ArchitectureElement* "Remote Desktop".

To sum up, the assistance is capable of dealing with different architecture abstraction levels. We have shown the possibility to enact the assistance process on a database that we have built. A prototype web application is implemented to show the feasibility of this security assistance.

6.3 A web application assistance prototype tool for architects

In order to evaluate the approach, we have developed the security assistance web application prototype tool to support the integration of security at the design phase and to be able to be used by architects.

The security assistance gives to non-security experts a lot of well-structured and motivated information on vulnerabilities and security controls, and is able to draw the attention to unknown security aspects for software system design and helps the architects to take more informed decisions.

The code source of the security assistance tool can be found here [1], and the web prototype application can be found here [2]. The web application performs server-side processing only, PHP (and Bootstrap for the front-end) is therefore the programming language chosen for the development of the application. The communication part with the database and its

1. https://github.com/lunanan/abs4sos
2. http://share-irisa.univ-ubs.fr/abs4sos/

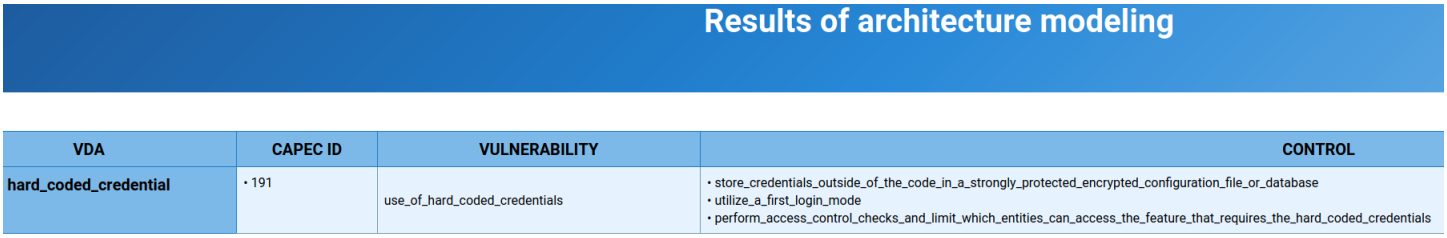
Figure 6.1 – The security assistance prototype tool home page

management are based on PostgreSQL and Sqlite3. PostgreSQL hosts the database, while Sqlite is required by the remote server manager. Sqlite is used within the web application to communicate with the server. Moreover, it is a solution for fast data maintenance, by performing updates, or some additions and/or deletions in a very light and therefore inexpensive way on the server side.

Figure 6.1 illustrates the home page of the security assistance prototype tool. As we can see, the tool requires the architect to select three types of information : an architecture element name, a security property and a category, which constitute an annotated DA. As a result, if the architect clicks on the "Domain Expert" button, the relevant security information is shown at the result page, such as the name of VDA, the vulnerabilities and the security controls, as shown in Figure 6.2. If the architect clicks on the "Security Expert tasks" button, he/she can view the details of each task of the assistance process (c.f. Figure 4.10).

It is important to specify that the the architect does not communicate directly with the database, he/she only selects the security requirement (including architecture element, security property and category) from the application's home page. The processing and conversion of the architect's requirements into SQL queries is performed by the server. In this way, it avoids some attacks such as "SQL Injection". The aesthetics of this web

application design will be improved in the future.



Results of architecture modeling

VDA	CAPEC ID	VULNERABILITY	CONTROL
hard_coded_credential	• 191	use_of_hard_coded_credentials	• store_credentials_outside_of_the_code_in_a_strongly_protected_encrypted_configuration_file_or_database • utilize_a_first_login_mode • perform_access_control_checks_and_limit_which_entities_can_access_the_feature_that_requires_the_hard_coded_credentials

Figure 6.2 – An example result of the security assistance prototype tool

6.4 Crossover experimentation

To evaluate the security assistance, we design a crossover study, which is widespread in software engineering experimentation [VAJ15]. A crossover design study is a longitudinal study in which each subject receives a sequence of different treatments. Treatments are applied by subjects, and subjects (people) are intrinsically quite different. Due to dissimilarities between people already existing prior to the experiment (competences, abilities, etc.), there may be a relatively large variability between different people applying the same treatment. For this reason, we choose the crossover experimentation. In this way, it controls the variability among subjects, who may conduct error-prone activity due to practice or fatigue. However, crossover experimentation may suffer some main criticisms: the carryover threat and its troublesome analysis. Carryover is the persistence of the effect of one treatment on another treatment which is applied later. To lighten this threat, we follow the good practice for the design and analysis of crossover study, proposed in [VAJ15].

To evaluate our security assistance tool, we compare its results with those of Microsoft SDL tool (available online [Cor18]), from both quality and quantity aspects. Microsoft SDL tool is implemented in the Microsoft SDL threat modeling process, which is based on STRIDE. Currently it is the most mature threat modeling method [She+18] and thus is also a security-by-design approach.

6.4.1 Crossover experimentation design

Sequence \ Period	Period 1	Period 2
Group 1: AB (Sequence 1)	Treatment A: Microsoft SDL tool (Case study 1)	Treatment B: security assistance (Case study 2)
Group 2: BA (Sequence 2)	Treatment B: security assistance (Case study 1)	Treatment A: Microsoft SDL tool (Case study 2)

Table 6.2 – AB-BA-Crossover Trial

The **goal** of this experiment is to investigate the effectiveness (the quality and the quantity of identified threats and vulnerabilities) of the two treatment tools: Microsoft SDL tool and our security assistance tool. The **null hypothesis** of the experiment is: *There is no difference of effectiveness between Microsoft SDL tool and our security assistance tool.*

The experiment has one **response variable**: the tool effectiveness. Tool effectiveness is measured with the quality and the quantity of threats (and of vulnerabilities) identified by subjects from case studies. The tool is the main experiment **factor**, with two **treatments**: Microsoft SDL tool and our security assistance tool.

The **subjects** participating in the empirical study are students from cyber software engineering school. They are knowledgeable in architecture design. We choose students with similar scores to decrease competences and abilities differences.

To conduct a crossover experiment, we follow the AB-BA-Crossover trial [Han17; VAJ15], which is illustrated in Table 6.2.

We ask all subjects to apply both tools. Therefore, there are two **periods** in the experiment (one to apply the first tool, and another to apply the second tool), as shown in Table 6.2. Each period takes place in a different session, held one week apart from one another. Each session lasts two hours.

We consider all possible sequences of the tool utilisation. Therefore, there are two experimental groups, each applying a different sequence. Subjects may be allowed to take a break between periods. As for two treatments, we provide two different case studies from different contexts, thus the carryover threat can be omitted. A prior we do not think that there is a chance of either of the sequences improving the experimental results of the other, as subjects are given different case studies as inputs for the tool utilisation.

For the case study 1, we use the one presented in the motivating example in Section 4.2. Subjects are not aware of this case study in advance. For the case study 2, we use one in the domain of running race, which has a similar scale than the case study 1, containing

both abstract and concrete architectural elements. The description of case study 2 is as follows:

> In a running race, RFID systems are used for precise timing. RFID tags help with the automation of the timestamp collection by reading player's tags and with related updates to real-time data in a central database. With this precise data, race coordinators do not need to manually record the times for each participant. Moreover, race statistics can be easily transferred to servers after authentication with a authentication protocol and made available via the Web. The RFID readers at the start and finish lines play a critical role in this scenario since they communicate with the participants' RFID tags as soon as participant leaves the start line and as soon as they enter the finish line. Moreover, to ensure that a runner follows a required route and not just show up at the start and finish points by taking short-cuts in-between, participant should wear a GPS receiver to trace the itinerary. The GPS receiver used in this scenario is a garmin gps map 60csx receiver.

The experimental procedure consists of six sessions. The first two two-hour sessions are training sessions in which subjects learn the tools' functionalities. The other two two-hour sessions are experiment execution periods. In each period, each group apply a tool to identify threats on the case studies. More threats number and better quality (pertinent threats) indicate greater effectiveness. The procedure is as follows: subjects apply the corresponding tool on the corresponding case study; afterwards, they identify with the corresponding tool the threats (and the vulnerabilities) on the case study.

6.4.2 Crossover experimentation results

The Table 6.3 shows the quantity results of this crossover experimentation. In the following, we discuss the details of this results with each group and each period.

6.4.2.1 Results of period 1 with group 1

During the period 1, the subject 1 in group 1 apply the Microsoft SDL tool on the case study 1 to identify threats. Figure 6.3 shows the Data Flow Diagram modeling the case study 1 by the subject 1 in group 1, and Figure 6.4 shows the result (the identified threats) generated by Microsoft SDL tool on the case study 1. As we can notice, the subject has modeled the architecture element the DA2 "database server", DA3 "web browser", and

	Period 1		Period 2	
Group 1:	*Treatment A Microsoft SDL tool (Case study 1)*		*Treatment B security assistance (Case study 2)*	
AB (Sequence 1)	Number of Threats	Number of Vulnerabilities	Number of Threats	Number of Vulnerabilities
Subject 1	4	X	24	52
Subject 4	15	X	21	49
Subject 5	15	X	28	70
Group 2:	*Treatment B security assistance (Case study 1)*		*Treatment A Microsoft SDL tool (Case study 2)*	
BA (Sequence 2)	Number of Threats	Number of Vulnerabilities	Number of Threats	Number of Vulnerabilities
Subject 2&3	10	39	14	X
Subject 6	16	12	6	X
Subject 7	16	9	7	X

Table 6.3 – Crossover study results

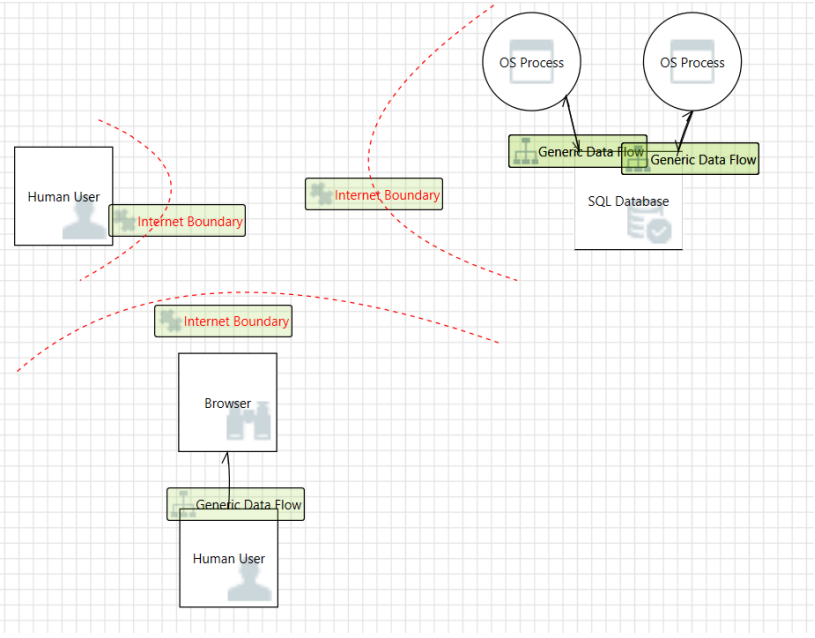


Figure 6.3 – Data flow diagram modeling case study 1 by group 1

ID	Diagram	Changed By	Last Modified	Title	Category	Description	Justification	Interaction	Priority
49	Diagram 1		Generated	Spoofing of D...	Spoofing	SQL Database...		Generic Data F...	High
50	Diagram 1		Generated	Potential SQL I...	Tampering	SQL injection i...		Generic Data F...	High
51	Diagram 1		Generated	Potential Exces...	Denial Of Servi...	Does OS Proce...		Generic Data F...	High
52	Diagram 1		Generated	Spoofing of S...	Spoofing	SQL Database...		Generic Data F...	High
53	Diagram 1		Generated	Weak Access C...	Information Di...	Improper data...		Generic Data F...	High
54	Diagram 1		Generated	Spoofing of S...	Spoofing	SQL Database...		Generic Data F...	High
55	Diagram 1		Generated	Weak Access C...	Information Di...	Improper data...		Generic Data F...	High
56	Diagram 1		Generated	Spoofing of D...	Spoofing	SQL Database...		Generic Data F...	High
57	Diagram 1		Generated	Potential SQL I...	Tampering	SQL injection i...		Generic Data F...	High
58	Diagram 1		Generated	Potential Exces...	Denial Of Servi...	Does OS Proce...		Generic Data F...	High

Figure 6.4 – Results of Microsoft SDL tool on the case study 1

architecture_element_container_name	architecture_element_component_name	vulnerability_name	threat_id_in_capec
medtronic 24950 mycarelink patient monitor	hard coded credential	use of hard coded credentials	191
medtronic 24950 mycarelink patient monitor	readable sensitive information	missing protection against hardware reverse engineering using integrated circuit imaging techniques	37
medtronic 24950 mycarelink patient monitor	readable sensitive information	debug power state transitions leak information	37
medtronic 24950 mycarelink patient monitor	readable sensitive information	improper scrubbing of sensitive data from decommissioned device	37
medtronic 24950 mycarelink patient monitor	readable sensitive information	sensitive information uncleared during hardware debug flows	37
medtronic 24950 mycarelink patient monitor	readable sensitive information	improper zeroization of hardware register	37
medtronic 24950 mycarelink patient monitor	readable sensitive information	cleartext storage of sensitive information in executable	37
medtronic 24950 mycarelink patient monitor	readable sensitive information	cleartext storage of sensitive information in a cookie	37
medtronic 24950 mycarelink patient monitor	readable sensitive information	cleartext storage in the registry	37
medtronic 24950 mycarelink patient monitor	readable sensitive information	cleartext storage of sensitive information	37
medtronic 24950 mycarelink patient monitor	readable sensitive information	use of web browser cache containing sensitive information	37
medtronic 24950 mycarelink patient monitor	readable sensitive information	missing encryption of sensitive data	37
medtronic 24950 mycarelink patient monitor	readable sensitive information	sensitive information uncleared in resource before release for reuse	37
medtronic 24950 mycarelink patient monitor	readable sensitive information	sensitive information uncleared during hardware debug flows	204
medtronic 24950 mycarelink patient monitor	readable sensitive information	improper zeroization of hardware register	204
medtronic 24950 mycarelink patient monitor	readable sensitive information	missing encryption of sensitive data	204
medtronic 24950 mycarelink patient monitor	readable sensitive information	use of cache containing sensitive information	204
medtronic 24950 mycarelink patient monitor	readable sensitive information	protection mechanism failure	57
medtronic 24950 mycarelink patient monitor	readable sensitive information	improper authentication	57
medtronic 24950 mycarelink patient monitor	readable sensitive information	channel accessible by non endpoint	57
medtronic 24950 mycarelink patient monitor	readable sensitive information	improper input vallidation	261
medtronic 24950 mycarelink patient monitor	readable sensitive information	insecure temporary file	155
medtronic 24950 mycarelink patient monitor	readable sensitive information	use of hard coded credentials	191
aerospike database server 3.10.0.3	untested sql input	improper enforcement of message or data structure	7
aerospike database server 3.10.0.3	untested sql input	incorrect comparison	7
aerospike database server 3.10.0.3	untested sql input	improper input vallidation	7
aerospike database server 3.10.0.3	untested sql input	improper neutralization of special elements in output used by a downstream component	7
aerospike database server 3.10.0.3	untested sql input	improper neutralization of special elements used in an sql command	7
aerospike database server 3.10.0.3	untested sql input	improper input vallidation	110
aerospike database server 3.10.0.3	untested sql input	improper neutralization of special elements used in an sql command	110
aerospike database server 3.10.0.3	untested sql input	process control	108
aerospike database server 3.10.0.3	untested sql input	improper neutralization of special elements used in an os command	108
aerospike database server 3.10.0.3	untested sql input	improper input vallidation	108
aerospike database server 3.10.0.3	untested sql input	improper neutralization of special elements in output used by a downstream component	108
aerospike database server 3.10.0.3	untested sql input	improper neutralization of special elements used in an sql command	108
aerospike database server 3.10.0.3	untested sql input	improper enforcement of message or data structure	66
aerospike database server 3.10.0.3	untested sql input	incorrect comparison	66
aerospike database server 3.10.0.3	untested sql input	improper input vallidation	66
aerospike database server 3.10.0.3	untested sql input	improper neutralization of special elements in output used by a downstream component	66
aerospike database server 3.10.0.3	untested sql input	improper neutralization of special elements used in an sql command	66

Figure 6.5 – Results of security assistance on the case study 1

"human user", which are abstract architecture elements. There are four types of threats generated by the Microsoft SDL tool, and no vulnerability and mitigation information is provided. The detail of threats generated by the Microsoft SDL tool and the result from other subjects in group 1 can be found in Appendix A.

6.4.2.2 Results of period 1 with group 2

During the period 1, the subjects 2 and 3 in group 2 apply the security assistance tool on the case study 1 to identify threats. Figure 6.5 shows the result generated from the security assistance database. The threats identified by the security assistance is indicated

vulnerability_name	security_control
use of hard coded credentials	store credentials outside of the code in a strongly protected encrypted configuration file or database
use of hard coded credentials	utilize a first login mode
use of hard coded credentials	perform access control checks and limit which entities can access the feature that requires the hard coded credentials
protection mechanism failure	protection of authentication mechanism
protection mechanism failure	enforce principle of least privilege
protection mechanism failure	harden registry server and file access permissions
protection mechanism failure	implement communications to and from the registry using secure protocols
improper input vallidation	use input validation before writing to a web log
improper input vallidation	validate all log data before it is output
improper input vallidation	strong input validation
improper input vallidation	use of parameterized queries or stored procedures
improper input vallidation	use of custom error pages
use of hard coded credentials	store credentials outside of the code in a strongly protected encrypted configuration file or database
use of hard coded credentials	utilize a first login mode
use of hard coded credentials	perform access control checks and limit which entities can access the feature that requires the hard coded credentials
improper enforcement of message or data structure	strong input validation
improper enforcement of message or data structure	use of parameterized queries or stored procedures
improper enforcement of message or data structure	use of custom error pages
incorrect comparison	strong input validation
incorrect comparison	use of parameterized queries or stored procedures
incorrect comparison	use of custom error pages
improper input vallidation	use input validation before writing to a web log
improper input vallidation	validate all log data before it is output
improper input vallidation	strong input validation
improper input vallidation	use of parameterized queries or stored procedures
improper input vallidation	use of custom error pages
improper neutralization of special elements in output used by a downstream component	enforce principle of least privilege
improper neutralization of special elements in output used by a downstream component	harden registry server and file access permissions
improper neutralization of special elements in output used by a downstream component	implement communications to and from the registry using secure protocols
improper neutralization of special elements in output used by a downstream component	strong input validation
improper neutralization of special elements in output used by a downstream component	use of parameterized queries or stored procedures
improper neutralization of special elements in output used by a downstream component	use of custom error pages
improper neutralization of special elements used in an sql command	strong input validation
improper neutralization of special elements used in an sql command	use of parameterized queries or stored procedures
improper neutralization of special elements used in an sql command	use of custom error pages
improper input vallidation	use input validation before writing to a web log
improper input vallidation	validate all log data before it is output
improper input vallidation	strong input validation
improper input vallidation	use of parameterized queries or stored procedures
improper input vallidation	use of custom error pages

Figure 6.6 – An excerpt of security control information generated by the security assistance on the case study 1

with “threat_id_in_capec”, which is the id of the attack pattern in CAPEC. The security assistance have identified 10 attack patterns and 39 vulnerabilities extracted from CWE. Figure 6.6 is an excerpt of the security control results identified by the security assistance basing on the mitigation information in CAPEC and CWE. The detail of threats generated by the security assistance and the result from other subjects in group 2 can be found in Appendix B.

6.4.2.3 Results of period 2 with group 1

During the period 2, the subject 1 in group 1 apply the security assistance on the case study 2 to identify threats. Figure 6.7 shows the security assistance results applying on the case study 2. 24 threats and 52 vulnerabilities are identified by the security assistance. The security controls can be therefore retrieved basing on the mitigation information from CAPEC and CWE. The detail of threats generated by the security assistance and the result from other subjects in group 1 can be found in Appendix C.

architecture_element_container_name	architecture_element_component_name	vulnerability_type_id	threat_id_in_capec
web_service	untested xml input	28	250
web_service	untested xml input	13	250
web_service	untested xml input	31	250
web_service	untested xml input	67	250
web_service	untested xml input	66	250
gps_receiver	modifiable gps signal	<null>	628
gps_receiver	modifiable gps signal	<null>	627
rfid_system	modifiable rfid tag	<null>	399
rfid_system	modifiable rfid tag	<null>	400
database_server	untested xml input	28	250
database_server	untested xml input	13	250
database_server	untested xml input	31	250
database_server	untested xml input	67	250
database_server	untested xml input	66	250
http_server	http	31	278
http_server	http	58	34
http_server	http	94	34
http_server	http	28	33
http_server	http	90	33
http_server	http	93	274
http_server	http	92	274
http_server	http	91	273
http_server	http	31	273
http_server	http	90	105
http_server	http	89	469
http_server	http	11	469
http_server	http	11	488
udp_server	udp	12	495
udp_server	udp	11	495
udp_server	udp	11	486
authentication_mechanism	manipulable authentication protocol	24	115
authentication_mechanism	manipulable authentication protocol	88	114
authentication_mechanism	manipulable authentication protocol	24	114
authentication_mechanism	manipulable authentication protocol	87	90
authentication_mechanism	manipulable authentication protocol	86	90
database_server	readable sensitive information	82	37
database_server	readable sensitive information	81	37
database_server	readable sensitive information	80	37
database_server	readable sensitive information	73	37
database_server	readable sensitive information	72	37
database_server	readable sensitive information	79	37
database_server	readable sensitive information	78	37
database_server	readable sensitive information	77	37
database_server	readable sensitive information	76	37
database_server	readable sensitive information	75	37
database_server	readable sensitive information	71	37
database_server	readable sensitive information	74	37
database_server	readable sensitive information	73	204
database_server	readable sensitive information	72	204
database_server	readable sensitive information	71	204
database_server	readable sensitive information	70	204
database_server	readable sensitive information	1	57
database_server	readable sensitive information	24	57
database_server	readable sensitive information	69	57
database_server	readable sensitive information	13	261
database_server	readable sensitive information	68	155
database_server	readable sensitive information	52	191

Figure 6.7 – Results of security assistance on the case study 2

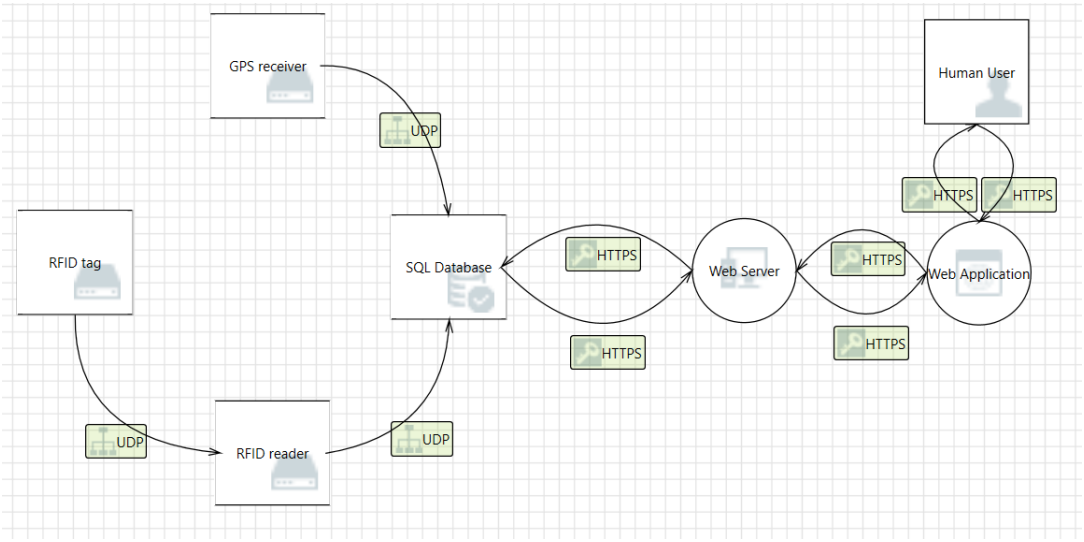


Figure 6.8 – Data flow diagram modeling case study 2 by group 2

6.4.2.4 Results of period 2 with group 2

During the period 2, the subjects 2 and 3 in group 2 apply the Microsoft SDL tool on the case study 2 to identify threats. Figure 6.8 shows the Data Flow Diagram modeling the case study 2 by subjects in group 2, and Figure 6.9 shows the result (the identified threats) generated by Microsoft SDL tool on the case study 2. There are 14 types of threats generated by the Microsoft SDL tool, and no vulnerability and security control information is provided. The detail of threats generated by the security assistance and the result from other subjects in group 2 can be found in Appendix D.

6.4.3 Crossover experimentation results analysis

In this section, we analyze the crossover experimentation results from both quality and quantity aspects. For the quality aspect, we check if the critical threats (identified by a security expert) on the common architecture elements that are dealt with by both tools are generated. For the quantity aspect, we apply the linear mixed model.

6.4.3.1 The quality analysis

For the case study 1, the common architecture element that is dealt with by both Microsoft SDL tool and our security assistance tool is the “Database Server”. For the

Threat List

ID	Diagram	Changed By	Last Modified	State	Title	Category	Description	Justification	Interaction	Priority
0	Diagram 1		Generated	Not Started	Spoofing of S...	Spoofing	SQL Database...		HTTPS	High
1	Diagram 1		Generated	Not Started	Cross Site Scri...	Tampering	The web server...		HTTPS	High
2	Diagram 1		Generated	Not Started	Persistent Cros...	Tampering	The web server...		HTTPS	High
3	Diagram 1		Generated	Not Started	Weak Access C...	Information Di...	Improper data...		HTTPS	High
8	Diagram 1		Generated	Not Started	Spoofing of S...	Spoofing	RFID tag may...		UDP	High
9	Diagram 1		Generated	Not Started	Spoofing of D...	Spoofing	RFID reader m...		UDP	High
10	Diagram 1		Generated	Not Started	Spoofing of S...	Spoofing	GPS receiver m...		UDP	High
11	Diagram 1		Generated	Not Started	Spoofing of D...	Spoofing	SQL Database...		UDP	High
12	Diagram 1		Generated	Not Started	Spoofing of S...	Spoofing	RFID reader m...		UDP	High
13	Diagram 1		Generated	Not Started	Spoofing of D...	Spoofing	SQL Database...		UDP	High
14	Diagram 1		Generated	Not Started	Web Server Pr...	Tampering	If Web Server i...		HTTPS	High
15	Diagram 1		Generated	Not Started	Cross Site Scri...	Tampering	The web server...		HTTPS	High
16	Diagram 1		Generated	Not Started	Elevation Usin...	Elevation Of Pr...	Web Applicati...		HTTPS	High
17	Diagram 1		Generated	Not Started	Spoofing the...	Spoofing	Human User m...		HTTPS	High
18	Diagram 1		Generated	Not Started	Cross Site Scri...	Tampering	The web server...		HTTPS	High
19	Diagram 1		Generated	Not Started	Elevation Usin...	Elevation Of Pr...	Web Applicati...		HTTPS	High
20	Diagram 1		Generated	Not Started	Cross Site Scri...	Tampering	The web server...		HTTPS	High
21	Diagram 1		Generated	Not Started	Elevation Usin...	Elevation Of Pr...	Web Server m...		HTTPS	High
22	Diagram 1		Generated	Not Started	Spoofing of D...	Spoofing	SQL Database...		HTTPS	High
23	Diagram 1		Generated	Not Started	Potential SQL I...	Tampering	SQL injection i...		HTTPS	High
24	Diagram 1		Generated	Not Started	Potential Exces...	Denial Of Servi...	Does Web Ser...		HTTPS	High

Figure 6.9 – Results of Microsoft SDL tool on the case study 2

"Database Server", at least the "SQL Injection" attack should be identified, which is one of the most critical attacks on "Database Server" (from a security expert's viewpoint). Both Microsoft SDL tool and our assistance tool have identified the "SQL Injection" attack. However, the Microsoft SDL tool presents only the threat information, while our assistance tool provide not only threat information, but also vulnerability and security control information to guide the architect.

For the case study 2, the common architecture elements that are treated by both Microsoft SDL tool and our security assistance tool are "Web Service", "Database Server", "UDP Protocol", "RFID Tag" and "GPS Receiver". For these architecture elements, at least the "XML Injection", "Cross Site Scripting", "SQL Injection", "Read Sensitive Data", "UDP Flood", "Data Tampering" attacks should be identified, which are considered as critical attacks for these element types from a security expert's viewpoint. As we can notice, our assistance tool has identified the above attacks (with both abstract and concrete attacks) except the "Cross Site Scripting" attack, which we believe is due to its prototype nature, and the Microsoft SDL tool has identified the above attacks except the "UDP Flood" attack. Therefore, both tools are capable of identifying main critical attacks, but both of them can not ensure their completeness. Nevertheless, our security assistance tool allows to provide vulnerability and security control information besides the threat information, while the Microsoft SDL tool does not, as shown in Table 6.4. It is worth noting that our security assistance tool does not include the human user as a vulnerable asset yet, while the Microsoft SDL tool does. This is because that at the

Capability / Tool	Identify critical threats	Threats reference	Identify vulnerabilities	Propose security controls
Microsoft SDL Tool	X	STRIDE		
Security Assistance	X	CAPEC/CWE (MITRE)	X	X

Table 6.4 – Tool quality comparison

current state, we have not completed the VA extraction from CAPEC, whereas CAPEC includes the social engineering attacks concerning to the human aspect. It's inclusion into our assistance is a matter of time.

6.4.3.2 The quantity analysis

As suggested in [VAJ15], if the period, sequence and carryover in a crossover experimentation have no influence, the paired t-test or the Wilcoxon test are possibilities to analyze the crossover experimentation. If they have influence, the linear mixed model is privileged. As we can not be sure that there is no influence of the period and sequence effect, we choose the linear mixed model in our crossover experimentation analysis, to take into the consideration the potential period and sequence effect.

The linear mixed model is the best method for analysing models with random coefficients and data dependency due to repeated measures [VAJ15]. The linear mixed model is a mixed-effect model, which is a statistical model containing both fixed effects and random effects, where random effects are often used to describe the subject-specific effect, while fixed effects are used to describe population-level effect.

Mixed effect models are particularly useful in contexts where repeated measurements are taken on the same statistical units, or where measurements are made on clusters of related statistical units [PC99]. The correlation between the repeated measurements is captured by the random effects and their distribution assumption [KJ07]. Furthermore, the mixed-effects model can handle missing and unbalanced data, which are common in practice, especially for longitudinal data analysis [PC99]. Therefore, mixed models deal with messy data and allow to use all available data, even when we have low sample sizes for structured data with many covariates to be fitted. Moreover, it allows to reserve degrees of freedom compared to standard regression.

Since the main purpose of the crossover study is to investigate treatment effects, we consider a linear mixed effect model which directly models the treatment effect [Zho12].

The model includes the following terms: tool(treatment), period and sequence as fixed factors, and subject as random factor nested within sequence, because subjects have been sampled from infinite population.

The variables are denoted as below: y_{ikj} is the observation for i^{th} sequence (i = 1,2), j^{th} subject (j = 1,2,....n), k^{th} period (k = 1,2). s is the random subject effect, which is assumed to be a random variable with mean zero and variance σ_s^2; e is a within-subject error, which is assumed to be random variable with mean zero and variance σ^2.

We directly estimate the parameters involved, and test whether there is treatment effect based on the results obtained from LME model. The model can be described by:

$y_{ijk}=\beta_0 + \beta_1 X_{treat} + \beta_2 X_{period} + \beta_{12} X_{treat} X_{period} + s_{ij} + \varepsilon_{ijk}$

s_{ij} is a random subject effect term and s_{ij} is assumed to be independently and identically distributed with N(0, σ_s^2); ε_{ijk} is error term and is assumed to be independently and identically distributed with N(0, σ^2).

Two dummy variables are introduced as:

$X_{treat} = 1$ if the subject is in treatment B (security assistance), and $X_{treat} = 0$ otherwise (Microsoft SDL tool);

$X_{period} = 1$ if the subject is in period 1, and $X_{period} = 2$ otherwise.

Introducing random subject effects in the regression model can capture the within-correlation of the subject observations. In addition, random subject effects could recover the information in the subject totals [Zho12]. However, if there is little variation in the subject totals, the between-subject variation will be small and the random effect may not need to be included. The variation can often be described by the intraclass correlation, which is defined as $\sigma_s^2/(\sigma_s^2 + \sigma^2)$, meaning the size of the between-subject variance (σ_s^2) relative to the within-subject variance (σ^2). In many crossover studies, the between-subject correlation is expected to be large and the remaining within-period correlation is expected to be small.

Table 6.5 summarized the results based on LME. As we can remark, although not statistically significant, the coefficient of the fixed factor treatment is positive, thus the treatment security assistance increases the discovered threats. Adding more data in the dataset to conduct this crossover experimentation is the perspective of our work.

Figure 6.10 and Figure 6.11 show the statistical result of the crossover experimentation.

Figure 6.10 shows that for case study 1, we do not see a significant difference, whereas for case study 2, the assistance identifies more threats than the Microsoft's one. Figure 6.11 shows the result of each experimenter. It can be seen that all participants have better

Random effects:			
Groups	Name	Variance	Std.Dev.
sequence:subject	(intercept)	1.780579	1.33438
subject	(intercept)	0.003372	0.05807
Residual		28.074977	5.29858
Number of obs: 14, groups: sequence:subject, 7; subject, 7			
Fixed effects:			
	Estimate	Std. Error	t value
(Intercept)	2.661	4.948	0.538
treatment	**7.875**	2.862	2.752
period	5.125	2.862	1.791
Correlation of Fixed Effects:			
		(Intr)	trtmnt
treatment		-0.413	
period		-0.909	0.143

Table 6.5 – Estimations of random and fixed effects components for linear mixed-effects model.

results with the assistance than the Microsoft SDL tool, except the subject 3.

6.4.4 Threats to validity

As illustrated below, in this crossover design, we have respected the good practices identified in [VAJ15], in order to lighten the crossover study internal and external threats.

For example:

— To counterbalance the "learning by practice and copying" threat that can affect periods, we provide two case studies from different contexts. These two case studies should have similar scale and complexity.

— To lighten the "history" threat that can affect periods, we have waited one week to conduct crossover experiment after training tasks.

— To avoid the "tiredness and/or boredom" threat that can affect periods, the session of each period should not be too long, in our case, each session last two hours. However, the changes that the subjects may experienced between two periods, are possible internal validity threats.

— To counterbalance the "carryover effect" threat, we provide two different case studies with similar scale and complexity for each treatment, thus the carryover effect in our study can be omitted.

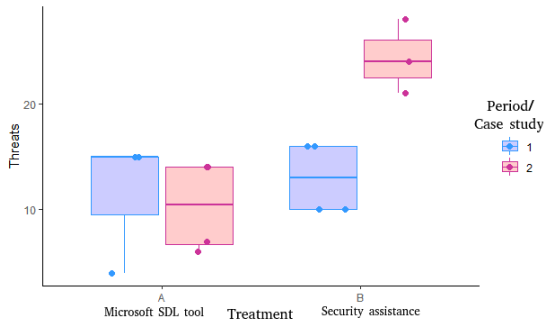


Figure 6.10 – The quantity results of threats identified in each period

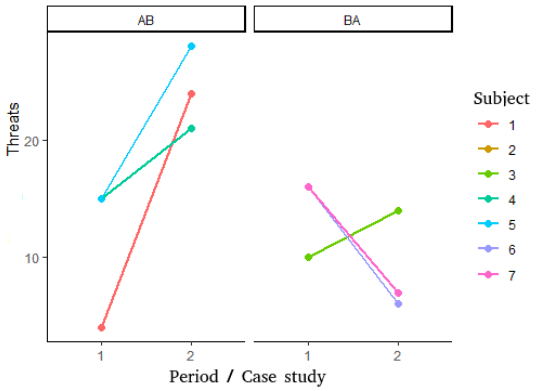


Figure 6.11 – The quantity results of threats identified by each subject

— The choice that the chosen sequences is two is because that there are no blocking variables. It is worth noting that the treatment-induced learning effect can be studied.

6.4.5 The usefulness evaluation – questionnaire

As we also aim at measuring the usefulness of the security assistance, after the effectiveness evaluation, we have also prepared a questionnaire concerning the easy-of-use of the security assistance. According to ISO 9126 [ISO01], usability is "the capability of the software product to be understood, learned, used and be attractive to the user, when used under specified conditions". The usability includes the sub-characteristics such as understandability, learnability, operability, attractiveness and usability compliance [Vli07].

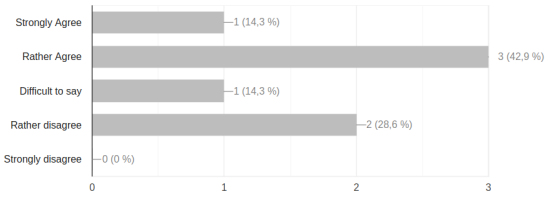


Figure 6.12 – Q1 result

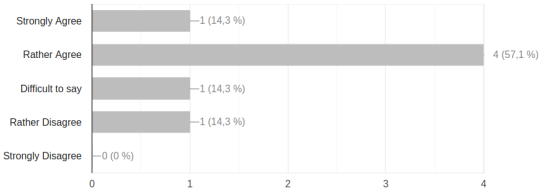


Figure 6.13 – Q2 result

At the current state, we have proposed four evaluation questions concerning to the operability, learnability and understandability sub-characteristics, and their results from the subjects are displayed in Figure 6.12, Figure 6.13, Figure 6.14 and Figure 6.15.

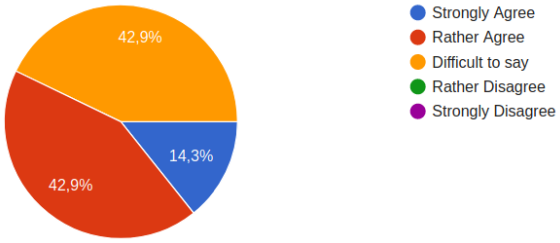


Figure 6.14 – Q3 result

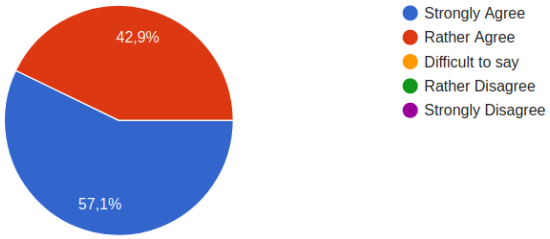


Figure 6.15 – Q4 result

— Q1: The assistance tool is easy to use (operability);

— Q2: The instruction for the assistance tool is easy to understand (learnability);

— Q3: The output of the tool is easy to use (operability);

— Q4: The output of the tool is easy to understand (understandability).

6.4.6 Experimentation results discussion

With the crossover experimentation results, we can observe that from the quality aspect, our security assistance tool identifies pertinent threats as well as the Microsoft SDL tool. The advantage of the security assistance tool is that it can also display the information about vulnerabilities and security controls, while the Microsoft SDL tool does not.

Therefore, to evaluate the initial null hypothesis of this crossover experimentation. We can conclude that for the quality aspect of the assistance's effectiveness, our security assistance provide more types of information (vulnerability and security control) than the Microsoft SDL tool. The quantity results show that the security assistance identifies in general more threats that Microsoft SDL tool, because the assistance's data reference is based on CAPEC and CWE, which contain more details than STRIDE, the reference used by Microsoft SDL tool.

As the aim of this work is to provide a security assistance to architects, the usability of this assistance is also an important quality characteristic to be satisfied.

For the usability, the tool should be able to hide as much as possible information not directly of interest for architects but, at the same time, it should give the right amount of information to take informed decisions and avoid other kind of risks [Gan+17]. It is worth noting, however, that the prototype nature of the tool may have affected such evaluation.

From the questionnaire results, we have observed that the security assistance is rather easy-to-use and the output of the security assistance is also rather easy-to-use. Moreover, the instruction for the assistance tool is rather easy to understand, and the output of the security assistance is easy to understand.

6.5 Conclusion

In this chapter, we have evaluated our first contribution (security assistance) in three ways: illustration of the feasibility of the security assistance's enactment by applying it on the motivating example, the proposition of a tool suite and the conduction of a crossover experimentation.

We have shown that the proposed security assistance is effective and useful for architects who may have limited security knowledge, thus can support the security-by-design approach. According to the evaluators' opinion, the tool is valuable, as it provides a lot of information on vulnerabilities, threats and security controls and helps even non-security experts to take more informed decisions related to security. However, a drawback is represented by the large amount of information that is generated and presented by the assistance tool, which may be partly overlooked or neglected by both security experts and non-security experts and that must be kept contained to maximize usability.

Chapter 7

Threat Modeling Proof-of-Concept

Contents

7.1 Introduction

In this chapter, we evaluation our second contribution by comparing our asset identification process with other current widely-used threat modeling processes, in order to verify that the models and the processes presented in Chapter 5 are useful. Next, we discuss the integration of our asset identification process into the current threat modeling process, such as the Microsoft one. We then discuss the advantages and the limitations of this integration. Then, we introduce a prototype BASH script for the VA extraction, which can be extended by security experts for further completion and reuse. Finally, we propose to evaluate our asset identification process with a crossover experimentation.

7.2 Threat modeling process illustration

In this section, we first illustrate a case study using the Microsoft SDL threat modeling process to enumerate threats. We choose Microsoft SDL threat modeling process because it is widely-used in the industry for threat modeling. As we will see, this process lacks of an "identifying asset" activity, which is a bridging step between the "domain modeling" and the "threat identification" activities. Therefore, we then illustrate the integration of our asset identification process into the Microsoft process as a complementary step to improve the detection of relevant threats.

7.2.1 Microsoft SDL threat modeling process

As to the case study, we take the example of "Web Sphere 7.0 application server", which is a software framework that hosts java-based web applications, allowing deploying and managing applications ranging from simple Web sites to powerful on-demand solutions. It is designed as a distributed computing platform that could be installed on multiple operating system instances, collectively referred to as a WebSphere cell. Its configuration information are tracked in XML configuration files throughout the cell.

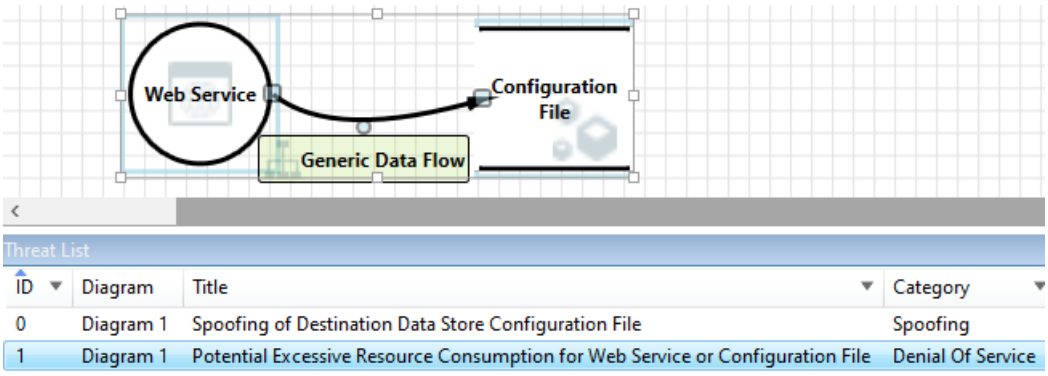


Figure 7.1 – Applying Microsoft SDL threat modeling tool

The Microsoft threat modeling process begins by characterizing the software or system ("Web sphere 7.0" in our case), by decomposing it and describing its components and data flows, using Data Flow Diagram (DFD), before enumerating threats by using "STRIDE per element" [Sho08]. There are four types of elements in DFD: *External Entity, Process, Data Flow and Data Store*. An excerpt of a possible decomposition of the "Web Sphere 7.0 application server" is presented in Figure 7.1, obtained using the Microsoft tool. It is

decomposed into a process called “web service” and a data store termed “configuration file”. “Web service” interacts with “configuration file” through a “general data flow”. The above three DFD elements are predefined by the tool. Further DFD modeling of the case study is not presented here for the sake of readability and space reasons.

The DFD is focused on modeling the flow of data through the system, and the analysis of the Microsoft SDL threat modeling tool is based on these flows. Therefore, to obtain a threat list, it is necessary to define/include at least a data flow in our model. A data flow needs at least two elements. That is way, additionally to the “configuration file”, we add a predefined element of the process type, a “web service”. We connect the two through a “generic data flow”.

The next phase is the threat enumeration, which is usually conducted in a brainstorming meeting guided by “STRIDE by elements”, supported by the threat list generated automatically by the tool. The threat list contains the threats that menace each DFD element or a group of DFD elements. As shown in Figure 7.1, the tool has found two threats concerning “Web Service”, “Configuration File” and their interaction: 1) spoofing of destination data store configuration file (belonging to the threat category of Spoofing) in Figure 7.2 and 2) potential excessive resource consumption for web service or configuration file (Denial of Service) in Figure 7.3.

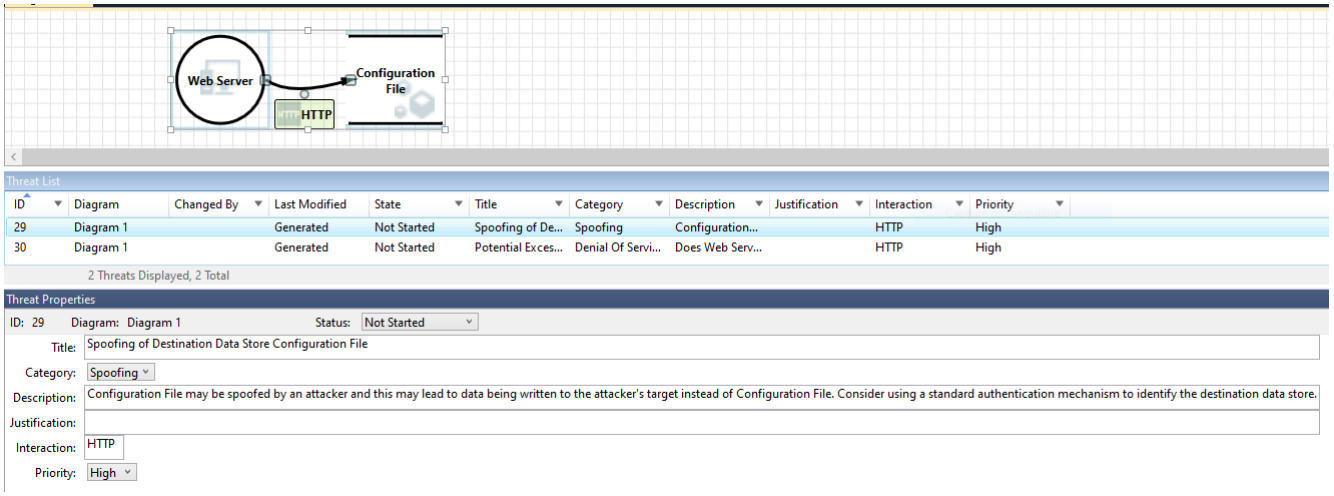


Figure 7.2 – Applying Microsoft SDL threat modeling tool

In the brainstorming meeting, participants generally discuss and find out potential threats belonging to six threat categories of STRIDE (*Spoofing, Tampering, Repudiation,*

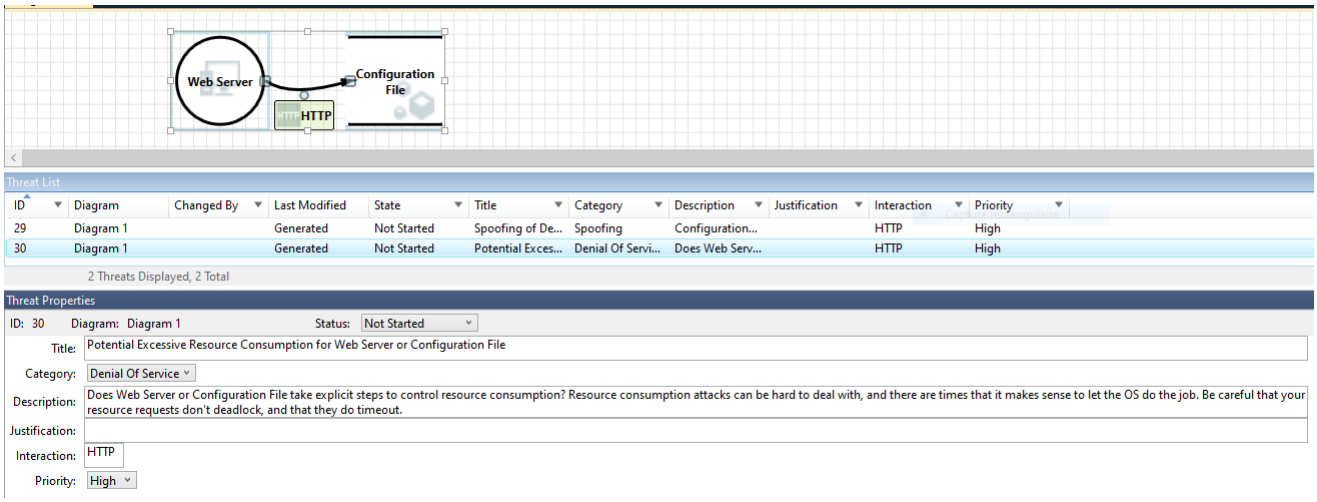


Figure 7.3 – Applying Microsoft SDL threat modeling tool

Information disclosure, Denial of service, Elevation of privilege) that can threaten the actual DFD element. At the current state of the work, we have not discussed with Microsoft SDL threat modeling experts. The quality and quantity of the results of the brainstorming meeting depends highly on participants. It is a highly subjective activity, the results of which are not reproducible. This makes it difficult for us to compare the brainstorming activity with our asset identification process. However, we believe that our process can help structure this activity, which we discuss in Section 7.3.

For the meeting without security experts, participants can use Microsoft SDL threat modeling tool to generate threats based on the DFD elements. As shown in Figure 7.2 and Figure 7.3, the results of threat enumeration about the category "deny of service" is not detailed and does not point out the root cause of the threat.

7.2.2 Integrating our process into Microsoft SDL threat modeling process

As we have mentioned, the Microsoft SDL threat modeling process begins by modeling the domain (by applying DFD), before identifying threats (by applying STRIDE). As we noted when discussing Table 3.1, the Microsoft SDL process does not contain the activity of "identifying asset", which is a bridging step between "modeling domain" and "identify threats". Therefore, we present the integration of our asset identification process into the Microsoft SDL threat modeling process in the aim of discovering more relevant threats.

Based on the DFD model in Figure 7.1, we observe that there is a loss of information during the domain modeling: the "configuration document" is of the XML type. This information may be critical for threat enumeration. A reason for this loss of information is that XML document is not predefined by the tool.

Therefore, we add the "XML document" in the domain model for our asset identification process, based on the DFD modeled in Figure 7.1, in order to fill the gap between domain modeling and threat enumeration. We describe the domain model of the case study using UML class diagram. Other modeling languages can be used as well. The domain asset model is presented in Figure 7.4. Conforming to the description of "Web Sphere 7.0 application server", the "Configuration Document" is of type XML and is contained in the "Web Sphere Server", together with "Web Service".

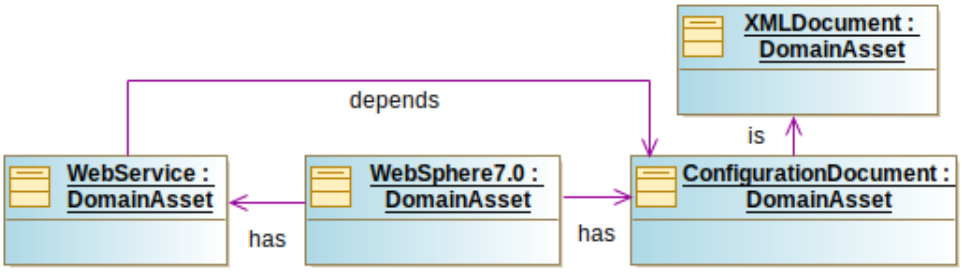


Figure 7.4 – An Excerpt of Domain Assets

The *WebApplicationServer* is modeled by a DFD containing (corresponding to the *has* relation in Figure 7.4 between *WebApplicationServer* and *XMLDocument*) a Web Service process and a Configuration File data store, which are related with a Generic Data Flow.

The Configuration file is equivalent to our *XMLDocument* and *ConfigurationDocument*. It is noteworthy that the Microsoft SDL tool does not manage different abstraction levels. Therefore, we can not model the *WebSphere7.0* and its specific *ConfigurationDocument*, together with their *has* relation. Moreover, we can not model the *is* relations between *Websphere7.0* and *WebApplicationServer*, and between *ConfigurationDocument* and *XMLDocument*.

To apply the asset identification process, on one hand, the domain experts produce the domain asset list, part of which is shown in Figure 7.4. The *WebSphere7.0* contains, among other components, a *ConfigurationDocument* and a *WebService*. These correspond to concrete (architecture) elements. The *ConfigurationDocument* can be generalized into an abstract (architecture) element *XMLDocument*.

On the other hand, the security experts use a vulnerable asset B-tree from the VA

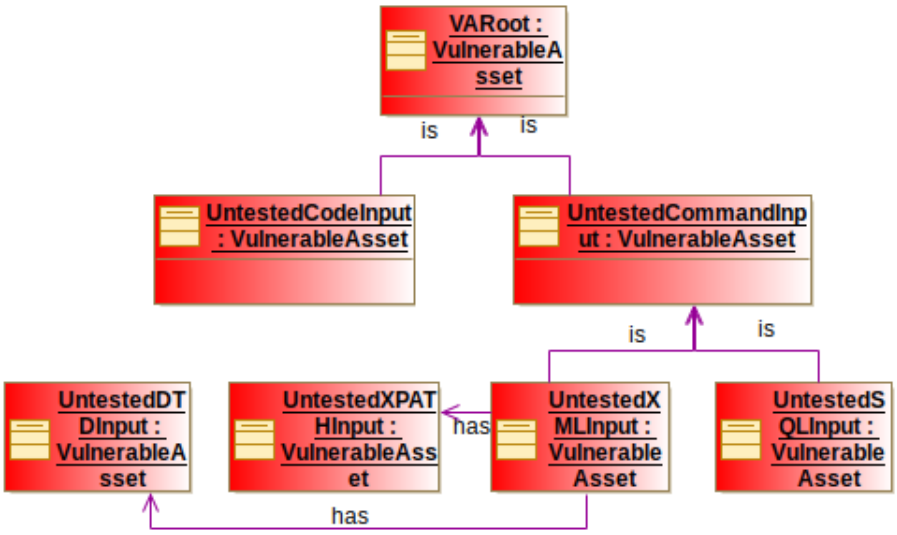


Figure 7.5 – An Excerpt of Vulnerable Asset Tree

library (constructed using the heuristic rules presented in Section 5.3), part of which is presented in Figure 7.5. This vulnerable asset B-tree begins by a root called *VARoot* (VA_1), which is an artificial root to start the process and can be specialized by any of its children VA, which contains a list of VA keys.

In Figure 7.5, the VA *UntestedCommandInput*, *UntestedCodeInput*, *UntestedXMLInput*, *UntestedSQLInput*, *UntestedDTDInput* and *UntestedXPATHInput* are respectively extracted: from the *meta* attack patterns *Command Injection* (CAPEC-248) and *Code Injection* (CAPEC-242); from the *standard* attack patterns *XML Injection* (CAPEC-250) and *SQL Injection* (CAPEC-66); and from the *detailed* attack patterns *DTD Injection* (CAPEC-228) and *XPATH Injection* (CAPEC-83), respecting the Rule 3 in Section 5.3. Other VA are omitted to assure the simplicity and the readability.

We illustrate the asset identification process based on the VA B-Tree in Figure 7.5. We initialize the process with i = 1, in this case, n is equal to 3. In the following, we illustrate each task of the process of Figure 5.3:

1) Select VA level i children: At the beginning of the process, i = 1, which is the *VARoot*. In our case, VA level 1 children are *UntestedCommandInput* (VA_{11}) and *UntestedCodeInput* (VA_{12});

2) Compare similarity: With the DA list provided by domain experts, security experts need to compare syntactical and semantic similarity between a vulnerable asset and a domain asset. Among the four domain assets *XMLDocument* (DA_1), *WebSphere7.0* (DA_2),

ConfigurationDocument (DA_3) and *WebService (DA_4)*, there is no similarity found when compared with *UntestededCommandInput (VA_{11})* and *UntestedCodeInput (VA_{12})*;

3) Therefore, for all DA_j, none of them is similar to VA_{1k}, which are children of VA level 1. In this case, i increments, now i is equal to 2, which is still lower than 3;

4) Select VA level i children: As no similarity is found from the upper level, the process advances to the lower level of the VA tree. For level i=2, there are two VA *UntestedCommandInput* and *UntestedCodeInput*, as shown in Figure 7.5. For the VA *UntestedCommandInput*, there are two children. Therefore, the VA level 2 children list contains *UntestedXMLInput* and *UntestedSQLInput*;

5) Compare similarity: *UntestedXMLInput (VA_{21})* is found both syntactically and semantically similar to *XMLDocument (DA_1)*;

6) Select VA_{ik} (VA_{21} in our case) keys: The process continues to search VA_{ik} keys. For our example, the VA_{21} key list contains *UntestedDTDInput* and *UntestedXPATHInput* (related to the *has* relation);

7) Select DA: Domain experts study if the domain model involves any of the assets which are in the VA_{21} key list, but have not yet been identified as domain assets. For the example in Figure 7.4, the domain experts realize that the *XMLDocument* DA does involve a *DTD*, which in this case can be manipulated by the user (attacker), without any intermediary tests. Possible impacts include that XML parsers, which process the DTD, consume excessive resources, resulting in resource depletion. Therefore, the *UntestedDTDInput* VA is a VDA that is vulnerable and involved in the domain;

8) Add into DA list: The domain experts add *UntestedDTDInput* to the DA list;

9) Add into VDA list: The security experts add as well, *UntestedDTDInput* to the VDA list. The VDA list for this example is shown in Figure 7.6. The same reasoning applies to other keys, which we do not detail here for the reason of readability;

10) After adding the discovered DA and VDA into each list, i increments, now i is equal to 3, which is not lower than n (=3 initially). Therefore, the process stops.

As a result of this process illustration, we have discovered the VDA *UntestedXMLInput* and *UntestedDTDInput*. *UntestedDTDInput* is not initially annotated as DA by domain experts. These two VDA are initially VA. Therefore, the threats that compromise these VA can be retrieved using the VA library presented in Section 5.3. As such, for *UntestedXMLInput*, the following 13 threats are identified:

— 1) XML Schema Poisoning (CAPEC-146);

— 2) XML Ping of the Death (CAPEC-147);

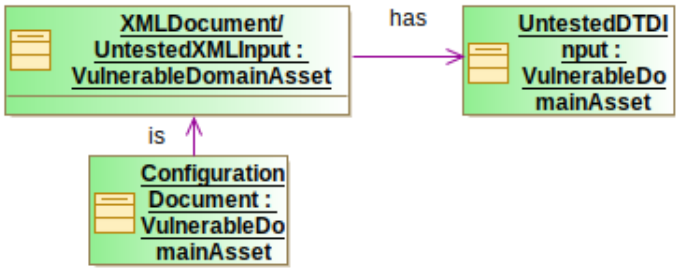


Figure 7.6 – An Excerpt of Vulnerable Domain Assets

— 3) XML Entity Expansion (CAPEC-197);
— 4) XML Entity Linking (CAPEC-201);
— 5) Spoofing of UDDI/ebXML Messages (CAPEC-218);
— 6) XML Routing Detour Attacks (CAPEC-219);
— 7) XML External Entities Blowup (CAPEC-221);
— 8) XML Attribute Blowup (CAPEC-229);
— 9) XML Nested Payloads (CAPEC-230);
— 10) XML Oversized Payloads (CAPEC-231);
— 11) XML Injection (CAPEC-250);
— 12) XML Quadratic Expansion (CAPEC-491);
— 13) XML Flood (CAPEC-528).

For *UntestedDTDInput*, there is only one threat identified: DTD Injection (CAPEC-228).

7.3 Discussion of threat modeling process

7.3.1 Case study discussion

Whereas the Microsoft SDL threat modeling tool has identified two threats for the case study, our asset identification process has identified 14.

Among these 14 threats, Spoofing of UDDI/ebXML Messages (CAPEC-218) and XML Routing Detour Attacks (CAPEC-219) belong to the same threat category of Spoofing, as the Spoofing of destination data store configuration file.

Similarly, XML Flood (CAPEC-528), XML Ping of the Death (CAPEC-147), XML Nested Payloads (CAPEC-230), XML Entity Expansion (CAPEC-197), XML Quadratic

Expansion (CAPEC-491), XML Oversized Payloads (CAPEC-231), XML Entity Linking (CAPEC-201), XML Attribute Blowup (CAPEC-229) and XML External Entities Blowup (CAPEC-221) belong to the same threat category Denial of Service, as the potential excessive resource consumption for web service or configuration file.

As we can see, we have identified more detailed threats comparing to the Microsoft SDL threat modeling tool. Moreover, our asset identification process has discovered new XML Schema Poisoning (CAPEC-146) and XML Injection (CAPEC-250) threats, which are not found by the Microsoft tool.

A number of threats identified by our process come from proposing VA as new DA, for example *UntestedDTDInput*. This enables identifying in-depth domain assets (that are also vulnerable), which otherwise may be overlooked.

Comparing to 14 threats discovered by our asset identification process, Microsoft SDL threat modeling tool has found only 2 threat categories, which need further discussion and clarification during the brainstorming meeting. The average number of overlooked threats is very high as mentioned in [SWJ13], which means that the catalog of 12 threat trees provided as checklist is not enough [HL06]. There is thus no guarantee that the brainstorming meeting can cover all 14 threats that we have found, because it depends highly on the security expertise, experiences and creativity of participants. The result of our asset identification process can thus be used as a checklist included in the brainstorming meeting to offer a guidance, to be complementary with Microsoft DFD and STRIDE based approach.

Moreover, the DFD is data-centric, it focuses on the data flow between components of the same abstraction level. Each new template, which can be created with the tool, can represent a new abstraction level. However, DFD does not allow presenting **relations** among elements of **different abstraction levels**. That is why we model the case study with UML class diagram, because it allows modeling elements with different abstractions levels, basing on the asset reference model presented in Figure 5.1.

To integrate DFD into our process, the four DFD element types can be mapped to our asset reference model. As shown in Figures 7.1 and Figure 7.4, the process “web service” is mapped into the *WebService* domain asset, the data store “configuration file” is mapped into the domain asset *ConfigurationDocument*, and the “general data flow” is mapped into *depends* relation to show the interaction. The DFD diagram containing these three elements is mapped into the domain asset *Websphere7.0* together with *has* relations.

A limitation of this case study would be that we have not compare our asset identifica-

tion process results with that of a real brainstorming meeting by industrial participants. This would be a perspective of this thesis.

7.3.2 General threat modeling discussion

As we have seen in Section 3.2.2.2, most of the existing threat modeling processes do not detail the asset identification phase. They usually consider it to be done through a discussion, usually of a non-structured, brainstorming type. The quality and quantity of the result of brainstorming meeting depends highly on participants. Moreover, such a discussion is highly creative and involves an important cognitive charge.

In our case, participants focus more on the category Spoofing and Denial of Service with the support of the tool. They may utilise other supports such as threat tree [HL06]. However, the quality and quantity of the result of threat enumeration depends highly on the security expertise, creativity and experiences of participants [Tor05].

By proposing a structured and detailed asset identification process together with the asset reference model, we help structure and guide this phase, which satisfies the Requirement 1 in Section 3.2.2.3. This asset identification process can be reused in different domains, as the VA library contains VA that is domain-independent. It is worth noting that several activities of our asset identification process still need human expertise, such as "similarity comparison" and "search if a VA is involved in the domain", these two human tasks pose yet a much easier cognitive load than that of the entire brainstorming, while allowing participants being creative in identifying threats.

Other problems encountered in non-structured brainstorming sessions are that some details or system parts are overlooked, or the stakeholder input is not captured accurately. Hence, more in-depth threat modeling is typically performed afterwards by a security expert in isolation, which can be error-prone, as it is performed by manually iterating through a model, and with a lack of specific domain knowledge, such as a particular technology used in the system [Ysk+20]. Our asset-based reference model can help consider both domain specific knowledge, by instantiating domain assets, and security knowledge, by extracting vulnerable assets. This two knowledge is shared by vulnerable domain assets (VDA), which can be established as a common vocabulary that can be understood by both experts, responding to the Requirement 3 in Section 3.2.2.3.

Therefore, the concept of *asset* is easily understandable by non-security experts compared to the concept of *threat* together with that of *attack technique*. Identifying VA that can later derive threats thus helps bridging the gap during the collaboration between

architects and security experts, satisfying Requirement 2 in Section 3.2.2.3.

7.4 A reusable BASH prototype for security experts

```
#Découper les mots simples et comparer avec cette liste de mots clés. (VA + mot clé)
extractKeyword1 <- function(){
  res <- c("Spoofing","Phishing","Hijacking","Overlay","Squatting",
           "Monitoring","Flood","Splitting","Smuggling","Tampering",
           "Bypass","Abuse","Overflow","Poisoning","Disabling",
           "Seizure","Jamming","Blocking","Alteration","Analysis",
           "Impersonation", "Manipulation", "Expansion", "Linking",
           "Blowup", "Fragmentation", "Misuse", "Exploitation",
           "Altered","Injection", "Pollution", "Inclusion",
           "Insertion", "Scanning", "Discovery", "Footprinting",
           "Fingerprinting", "Probe")
  return(res)
}
```

Figure 7.7 – Keywords in Cluster 1

```
#Ne pas découper le mots composés pour extractKeyword2,
#seulement localiser en text mining. (mot clé + VA)
extractKeyword2 <- function(){
  res <- c("Spoofing of ","Exploiting Incorrectly","Modification of",
           "Collect Data from","Pretexting via","Bypassing of ")
  return(res)
}
```

Figure 7.8 – Keywords in Cluster 2

To help enrich the VA library, we have written a BASH application prototype based on R language to extract automatically vulnerable assets [1], basing on the VA extraction rules presented in Section 5.3. The objective of this BASH application is to extract automatically vulnerable assets from attack patterns enumerated in CAPEC, by respecting the extraction rules presented in Section 5.3. This application allows to obtain a list of VA by either displaying the extraction results directly on the terminal, or record them into a CSV file (Excel) or a text document, as shown in Figure 7.10. To allow this extraction, we

1. https://github.com/lunanan/ArchwareExtraction

```
#Découper les mots simple et comparer avec cette liste de mots clés. (mot clé + VA)
extractKeyword3 <- function(){
  res <- c("Manipulate","Leveraging","Manipulating","Disabling","Accessing",
           "Intercepting","Modifying","Counterfeit","Fake the", "Exploit",
           "Using", "Leverage", "Bypassing", "Poison", "Infected", "Contaminate",
           "Detect","Probe", "Capture", "Sniffing")
  return(res)
}
```

Figure 7.9 – Keywords in Cluster 3

have identified three clusters of keywords in this prototype script, as shown respectively in Figure 7.7, Figure 7.8 and Figure 7.9, based on the rules presented in Section 5.3. Security experts can extend these clusters by adding more keywords or by proposing new rules that can help extract vulnerable assets.

The advantages of this application is firstly to bring a reusable VA extraction process for security experts, but also to propose an evolutive application by making it open source. In the future, we aim at automating this extraction process basing on more techniques such as text mining.

KEYWORD	VULNERABILITY_ASSET	CAPEC_ID
Manipulate	Registry Information	203
Manipulate	Human Behavior	416
Manipulate	Timing and State	172
Manipulate	Data Structures	255
Manipulate	System Resources	262
Leveraging	Race Conditions	26
Leveraging	Race Conditions via Symbolic Links	27

Figure 7.10 – An excerpt of BASH application result

7.5 Crossover experimentation

To evaluate our asset identification process that can be integrated during the brainstorming sessions in threat modeling, we plan to conduct also a crossover experimentation. The design and the analysis of this crossover experimentation is similar to the one presented in Section 6.4, with the following differences:

— Each **subject** in this crossover experimentation is a group of people who are participants during the brainstorming session, instead of a single subject in Section 6.4;

— The experiment has one **response variable**: the process effectiveness (instead of the tool effectiveness in Section 6.4);

Period / Sequence	Period 1	Period 2
Group 1: AB (Sequence 1)	Treatment A: Microsoft SDL threat modeling process (Case study 1)	Treatment B: integration with asset identification process (Case study 2)
Group 2: BA (Sequence 2)	Treatment B: integration with asset identification process (Case study 1)	Treatment A: Microsoft SDL threat modeling process (Case study 2)

Table 7.1 – Crossover experimentation for asset identification process evaluation

— The process is the main experiment **factor**, as shown in Table 7.1, with two **treatments**: Microsoft SDL threat modeling process and its integration with our asset identification process, instead of the Microsoft SDL tool and the security assistance tool in Section 6.4.

This crossover experimentation will be a part of a system security course, and will be conducted with 30 students who play different roles in the brainstorming session of threat modeling. However, due to the logistics constraint, this experimentation can only take place at January 11, 18 and 25, 2021, the result of which will not be included in this thesis, but rather in a journal paper.

7.6 Conclusion

In this chapter, we have given a proof-of-concept of our second contribution: structuring the asset identification phase in the threat modeling process, to bridge the gap between threat modeling participants (including architects and security experts) from the security expert's viewpoint. We have illustrated the widely-used threat modeling approach Microsoft SDL threat modeling tool with a case study, and we have integrated our asset identification process into the Microsoft one to show its improvement. A reusable open-source BASH application prototype has also been implemented to allow security experts extracting vulnerable assets basing on different clusters of relevant keywords. This prototype will be improved in the future by using more techniques such as text mining. To evaluate our asset identification process, we plan to conduct a crossover experimentation, similar to the one presented in Section 6.4. Due to the logistics constraint, we can only perform this crossover experimentation at January 11, 18 and 25, 2021, with 30 students who play different roles in the brainstorming session in threat modeling. Conducting this

crossover experimentation with real industry participants in a real brainstorming session of threat modeling in the industry will be the perspective of our work.

Part IV

General Conclusion

Chapter 8

General Conclusion

Contents

In this conclusion chapter, we firstly synthesize our contributions that fulfill the research objectives. Then, we discuss the results and their limitations. Based on our contributions together with their limitations, we finally present several perspectives and a number of possible future directions that would advance our research. Some of these perspectives are motivated by ongoing research works with other research teams in parallel with this PhD thesis.

8.1 Fulfilling the objectives and contributions

The wide adoption of modern software development methodologies is dramatically reducing the time-to-market of many innovative services. In this context, security has barely

been considered as a primary requirement to be taken into consideration from the early stages of software development lifecycle (SDLC). Part of the difficulty of security-critical systems development is that correctness is often in conflict with cost. While thorough methods of system design imply high cost through personnel training and use, they are all too often avoided [Jür02].

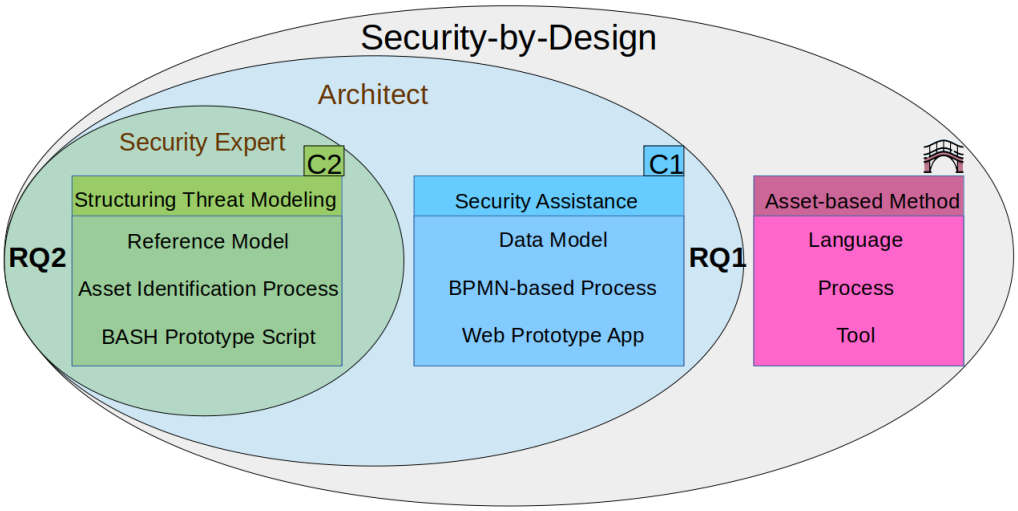


Figure 8.1 – High level view of the contribution

In this context, the collaboration between the architects and security experts is essential to ensure security at the design phase. However, this collaboration is often done through brainstorming sessions, with a significant gap in the understanding of the system and security concepts. To encounter this problem, in this thesis, we contribute to bridge the gap between architects and security experts, by taking a step closer from the architect' viewpoint and from the security expert's viewpoint, as shown in Figure 8.1. To do that, from the architects' viewpoint, we advocate the definition of a security assistance for the architect when designing secure software systems. At this aim, we designed and implemented a comprehensive data model to collect security-related information (assets, threats, vulnerabilities, etc.), basing on a novel refinement of the concept of "asset", to bridge the gap between architecture design and security concerns. The idea of this assistance is illustrated in an asset-based three-view security assistance framework, which includes: i) a data model and ii) a BPMN-based process.

To enact this security assistance, we have to collect security information as much as possible. For this, we rely on sound existing security knowledge repositories, such as CAPEC and CWE, to build the assistance database implementing the data model. We

rely on CAPEC and CWE because they are widely-used security repositories in the security community, and are updated frequently and are maintained dynamically by security experts. CAPEC can be used by people who understand security attack and defense. For those who are not security experts, they may have the difficulty to use it efficiently and effectively. We extract the essence Vulnerable Assets (VA) from CAPEC, and we provide a structured process to link VA with domain assets that can be understood by non-security experts. As a result, it can be used to provide the security assistance to help non security expert to retrieve security information.

To evaluate this security assistance, we firstly show that it is possible to automate it, by applying it on a telemonitoring system motivating example. This application highlights how the proposed assistance can help the architect when integrating the security aspect at the design stage. Then, a web application prototype tool has been implemented to show the enactment of this assistance. In addition, we have conducted a crossover experimentation to compare this security assistance tool with another widely-used security-by-design tool: the Microsoft SDL tool.

It is worth pointing out that the goal of our methodology is to help architects, especially those who have limited security skills, to identify, as early as possible, the main threats and vulnerabilities affecting the software system under design, in compliance with the principles of security-by-design.

Another contribution of this thesis is bridging the gap from the security expert's viewpoint. It aims at structuring the asset identification phase in the threat modeling process, as threat modeling is an essential approach that can integrate the security aspect at the design phase. Threat modeling is a result of a collaborative process involving many actors from different backgrounds. Despite its importance, the collaboration between architects and security experts to bridge the gap between domain modeling and threat enumeration phases is not trivial. One of the main reasons is that threat identification and enumeration is often a challenging task for non-security experts. Thus, architects who have limited security knowledge have to rely on threat modeling processes, which may quickly turn into a complex task when these processes lack guidance and formalisation.

To address this limitation, we have proposed an asset-based reference model and a systematic asset identification process to facilitate the collaboration between different actors. As a result, pertinent assets such as Vulnerable Assets are structured and Vulnerable Domain Assets are identified to improve the threat enumeration phase. Then, we have discussed how the proposed approach could be applied to structure the security knowledge

base (CAPEC) and how the proposed process could be integrated with, and complementary to, the Microsoft SDL threat modeling process, using an appropriate case study. Results show the usefulness of our findings in identifying new assets and threats, and in bridging the gap between the architects and the security experts through the formalisation of the brainstorming activity.

Our approach thus bridges the gap between architects and security experts. The Domain Asset (DA) can be represented by architecture elements. With the asset identification process, Vulnerable Domain Asset (VDA) can be integrated into the architecture model as well, allowing designing secure systems with VDA's controls. Our approach integrates into the architecture model, the security knowledge about possible attacks and vulnerabilities. Based on this knowledge, our approach proposes possible attack mitigations and/or security controls to the architect.

Moreover, one of the originality of our approach is that it can deal with the security aspect for both abstract and concrete architecture elements. The level of details and the relevance of the security controls depend on the abstraction level of the elements in the architecture model. The controls are more precise if the architecture elements are more concrete, whose vulnerabilities are cataloged in common security knowledge repositories. Indeed, the definition of the architecture of a software system can be performed in several periods and therefore it involves different levels of abstraction. It is thus typical to mix abstract elements (whose implementation details are not specific yet) with concrete ones (which are precise) during this activity. The architecture is thus refined gradually to become finally a concrete architecture model. Our asset reference model and the assistance data model response to this needs (with *is* (generalization) relation). In our approach, we model not only the different abstraction levels of general element types (thus potential architecture elements), but also different abstraction levels of attack patterns (in CAPEC), which can mirror the refinement of architecture elements (with the refinement of VA extracted from attack patterns).

8.2 Discussions and limitations

As discussed in the thesis, addressing security at the early stages of the SDLC, in particular at the design phase, in the modern software development methodologies is quite complicated, mainly due to the difficulties in making proper security design choices. The proposed methodology, along with the associated data models and prototyping tools,

aim at supporting architects, those who have few security skills.

The proof-of-concept evaluations have showed that the methodology is able to correctly identify a set of vulnerable domain assets, together with its vulnerability types and control types, but it does not provide any means to prove the completeness of the set of identified vulnerable domain assets. The effectiveness of the methodology naturally depends on the completeness of the VA library, which is highly dependent on the security repositories that we use to extract these VAs. Our security assistance knowledge base would be pertinent only if the security repositories that we rely on is pertinent. Therefore, it is fundamental that our assistance security knowledge base is continually updated to include more vulnerabilities and controls, and take into account a wider set of assets. That is why we choose CAPEC and CWE to extract their security information, because they are referential repositories that are updated frequently and dynamically by the security community. However, although natural languages provide freedom of expressiveness, they introduce challenges for analysis of attack patterns. Even if CAPEC and CWE are well organized, they still present the security information in the form of natural language, which poses a challenge for our VA extraction. We are still lacking a formal way to represent a complete attack in an unambiguous way. At the current state of this thesis, the VA extraction process is semi-automatic, which may be error-prone due to human mistakes and incomplete.

Another limitation is that our methodology currently addresses only technical assets, while organizational and human-related assets have not been taken into account and are out of the scope of this thesis, although we recognize that these may introduce additional unpredictable risks, such as social engineering attacks.

8.3 Perspectives

There are several perspectives for this PhD thesis, which we plan to continue as future works. We choose to present it under two angles: from different periods of time and from different topics.

8.3.1 In the short term

In the short term, namely the next coming months, we plan to improve the security assistance database and evaluate the security assistance with other domain applications.

8.3.1.1 Improve the security assistance database

To improve the security assistance database, we aim at automating the security knowledge extraction (e.g. VA extraction) from existing security repositories such as CAPEC, ATT&CK [MITa] and OWASP [Fou17], and its insertion into the assistance database, to reduce the load of human analysts and spare them valuable time to investigate more sophisticated, unknown attacks. Possible directions include text mining approaches such as topic modeling [Ada+18].

8.3.1.2 Evaluate the security assistance with other domain applications

In the short term, we we are also planning to evaluate the security assistance with other domain applications, especially, cloud domains. This is an ongoing work with members of DiverSE team in IRISA, Rennes, France. For now, we are identifying cloud domain assets, especially those which are also cloud vulnerable domain assets, basing on our VA library which we still complete.

8.3.1.3 Interdependence among assets

We are intending to take into consideration the inter-dependencies among domain assets and analyze the consistency of the architect's annotations (a security property and a category on a domain asset), using for example colored graphs [CR18] to identify if there are contradictions of the security property annotations on domain assets. This is an ongoing collaboration work with a professor from Tampere University.

8.3.2 In the medium term

Once we have automated the security knowledge extraction and have evaluated the security assistance with other domain applications, we have several medium-term objectives to improve the current PhD work.

8.3.2.1 Improve the security assistance database

To improve the security assistance database in the medium term, we plan to add other security-related concepts and mechanisms such as *Risk* and its computation, for which at least partial information could be extracted from existing bases, such as risk score

from Common Vulnerability Scoring System Calculator (CVSS) [1] [MSR07] and Common Weakness Scoring System (CWSS) [2].

We also plan to extend concepts such as *SecurityProperty* (e.g., to take into account authenticity, traceability, and privacy [Anc+19], etc.), *RelationType* (e.g., with XOR or NOT). This requires taking into account attack impacts explored from other existing repositories (e.g., ATT&CK, OWASP, etc.).

8.3.2.2 Formalization of element types

The use of CAPEC and CWE repositories to propose a security assistance uncovers a constraint on the Architecture Description Language (ADL) and on the way how this ADL is used. Indeed, it should be feasible to associate each architectural element with a "type" that can be referred to an "element" in the CPE repository. In fact, element types can be referred with a certain vocabulary (VA) in the security community, and they can also be mentioned by architects in their own vocabulary (DA). At a certain level, there will be a language vocabulary rupture for the matching activity (which is at the current of time a manual activity). It will be much more easier to manage this matching activity if the ADL uses the type mechanism to annotate the involved elements. This could be tackled for example via the expression of typing, refinement or composition relationships, compatible with CPE if the ADL offers such language mechanisms; or if not, at least by naming convention rules on architectural elements. It can thus make machine interpretation and processing possible.

To formalize and structure element types in the medium term, we plan to investigate alternatives to the current standard CPE-based naming scheme which relies on its appropriate application by the architect. These alternatives need to be transparent for the architect, which is a non-trivial task. Directions include: ADL typing mechanisms, ontology matching and text mining [Cun02].

We also plan to develop a tool suite which enacts the assistance during the architecture modeling phase, relying on SysML for example. Indeed, the SysML is designed to be easily extended, using for instance the "profile" mechanism or by adding new properties in blocks.

1. https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator
2. https://cwe.mitre.org/cwss/cwss$_v$1.0.1.*html*

8.3.3 In the long term

Finally, we also have some ultimate objectives as extensions of this PhD thesis.

8.3.3.1 Improve the security assistance database

To improve the security assistance database, we are planning to extend the VA library in order to include organizational and human-related vulnerable assets. For this, one option is to rely on the works relating to social engineering attacks to extract VAs of these attacks, and inspiring works about how we trace between high organizational level asset into technical asset such as the one proposed in [Der+18].

8.3.3.2 Coupling with Attack Trees

Annotating the nodes in attack trees with the information of asset impacted by each attack action is our another ultimate objective. Attack trees [Sch99] aim at modeling security threats by focusing on the attackers' motivations and actions to attack systems. Attack trees provide the possibility of adding more information to nodes, e.g. in order to indicate if the attack is possible/impossible or assign costs to each leaf node [TJR10]. We can also add the asset information on each node to show the impacted asset for each adversary action, in order to help domain experts understand better the domain elements impacted by each adversary action. The information of part of attack trees concerning a vulnerable asset can also be reused for different domains. Once the vulnerable asset is identified, attack tree that shows attacks against this vulnerable asset can be reused for all systems that involve this vulnerable asset.

Publication and Activity

Publications

This section lists the publications from October 2017 until now.

— Nan Messe, Nicolas Belloir, Vanea Chiprianov, Imane Cherfa, Régis Fleurquin and Salah Sadou. Development of Secure Systems of Systems Needing a Rapid Development, in Proceedings of the 14th annual System of Systems Engineering Conference (**SoSE** 2019), pages 152-157, Anchorage, USA. May 19-22, 2019. IEEE.

— Nan Messe, Nicolas Belloir, Vanea Chiprianov, Imane Cherfa, Régis Fleurquin and Salah Sadou. Le Développement de Système de Systèmes Sécurisé Nécessitant un Déploiement Rapide, in 8ème Conférence en IngénieriE du Logiciel (CIEL 2019), Toulouse, France.

— Nan Messe, Nicolas Belloir, Vanea Chiprianov, Jamal El-Hachem, Régis Fleurquin and Salah Sadou. An asset-based assistance for secure by design. 2020 27th Asia-Pacific Software Engineering Conference (**APSEC**), 2020. **Core rank: B**

— Nan Messe, Vanea Chiprianov, Nicolas Belloir, Jamal El-Hachem, Régis Fleurquin and Salah Sadou. Asset-oriented threat modeling. 2020 19th IEEE International Conference On Trust, Security And Privacy In Computing And Communications (**TrustCom**), 2020. **Core rank: A**

Besides, a journal paper including the work of crossover experimentation is planned to be submitted.

Professional activities

During this thesis period, I have also participated several activities to enrich my research life.

— **Services to the research community:**

  — **Presentation**

- Presentation of the paper “Asset-Oriented Threat Modeling” in TrustCom 2020 (01/2021);
- Presentation of the paper “An asset-based assistance for secure by design” in APSEC 2020 (12/2020)
- “GT Vélocité Logicielle” Day (12/2020)
- DiverSE Coffee presentation in DiverSE groupe of IRISA, Rennes (10/2020)
- RIMEL (Software Evolution) Day (04/2019)

- **Poster presentation**
  - GDR GPL day (06/2018)
  - International Symposium on Engineering Secure Software and Systems (ESSOS 2018) (06/2018)
- **Student volunteer**
  - ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS) (10/2018)

- **Scientific vulgarisation:**
  - **Competition**
    - MyThesis 3.0 Challenge (The 3rd European Cyber Week) (11/2018)
  - **Vulgarisation**
    - Digital Career’s Day (01/2020)
    - Requirements Engineering and Software Engineering for Cyber-Physical Systems’s Day (01/2019)
    - Digital Career’s Day (01/2019)
- **Other activities:**
  - **Participation of mentoring program as a mentee**
    - Mentor: David Pichardie (2019,2020)
    - Mentor: Patrice Quinton (2018)
  - **Summer school**
    - 6th International School on Software Engineering (07/2020)
    - Cyber in Occitanie (GDR Security) (07/2018)

- Requirements Engineering in agile and data-driven development contexts (Genova, Italy, 05/2018)
- IoT (CNRS) (12/2017)

— **Training course**

- SIF/Specif Campus Day
  Academic and Industrial Careers - What to do with a PhD in Computer Science? (12/2018)
- Preparation of My Thesis 3.0 challenge (11/2018)
- Creating DSL with Eclipse (SED) (11/2018)
- Scientifique watch (MATHSTIC) (03/2018)
- French (MATHSTIC) (02/2018 - 04/2018)
- Scientific integrity awareness training (MATHSTIC) (03/2018)
- Mendeley (MATHSTIC) (02/2018)
- Knowing higher education: its context, its students (MATHSTIC) (02/2018)
- Zotero (MATHSTIC) (12/2017)
- Introduction of research ethics (MATHSTIC) (10/2017)
- GIT tutorial (MATHSTIC) (11/2017)

BIBLIOGRAPHY

[AB15] Adel Alshamrani and Abdullah Bahattab. « A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model ». In: *International Journal of Computer Science Issues (IJCSI)* 12.1 (2015).

[AD02] Christopher Alberts and Audrey Dorofee. *Managing Information Security Risks: The OCTAVE Approach*. Addison-Wesley Professional, 2002.

[Ada+18] Stephen Adams, Bryan Carter, Cody Fleming, and Peter A Beling. « Selecting system specific cybersecurity attack patterns using topic modeling ». In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE. 2018, pp. 490–497.

[Alb+03] Christopher J. Alberts, Audrey J. Dorofee, James Stevens, and Carol Woody. *Introduction to the OCTAVE Approach*. Tech. rep. Software Engineering Institute, 2003. URL: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=51546.

[Alp+19] Sascha Alpers, Roman Pilipchuk, Andreas Oberweis, and Ralf Reussner. « The Current State of the Holistic Privacy and Security Modelling Approach in Business Process and Software Architecture Modelling ». In: *Information Systems Security and Privacy*. Ed. by Paolo Mori, Steven Furnell, and Olivier Camp. Cham: Springer, 2019, pp. 109–124. ISBN: 978-3-030-25109-3.

[Ame+12] David Ameller, Claudia Ayala, Xavier Franch, and Jordi Cabot. « How do Software Architects Consider Non-Functional Requirements: An Exploratory Study ». In: *2012 20th IEEE International Requirements Engineering Conference, RE 2012 - Proceedings*. Sept. 2012, pp. 41–50. DOI: 10.1109/RE.2012.6345838.

[Anc+19] Nicolas Anciaux, Philippe Bonnet, Luc Bouganim, Benjamin Nguyen, Philippe Pucheral, Iulian Sandu Popa, and Guillaume Scerri. « Personal Data Management Systems: The security and functionality standpoint ». In: *Information Systems* 80 (2019), pp. 13 –35. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2018.09.002. URL: http://www.sciencedirect.com/science/article/pii/S0306437918304022.

[ANS16] ANSSI. *EBIOS - Expression des Besoins et Identification des Objectifs de Sécurité*. 2016.

[Bed+13] Punam Bedi, Vandana Gandotra, Archana Singhal, Himanshi Narang, and Sumit Sharma. « Threat-oriented security framework in risk management using multiagent system ». In: *Software: Practice and Experience* 43 (Sept. 2013). DOI: 10.1002/spe.2133.

[BF12] Anh L Bui and Gregg C Fonarow. « Home monitoring for heart failure management ». In: *Journal of the American College of Cardiology* 59.2 (2012), pp. 97–104.

[BFM04] Eric J. Byres, Matthew Franz, and Darrin Miller. « The use of attack trees in assessing vulnerabilities in scada systems ». In: *in IEEE Conf. International Infrastructure Survivability Workshop (IISW '04). Institute for Electrical and Electronics Engineers*. 2004.

[BHH13] Kristian Beckers, Denis Hatebur, and Maritta Heisel. « A Problem-Based Threat Analysis in Compliance with Common Criteria ». In: *Proceedings - 2013 International Conference on Availability, Reliability and Security, ARES 2013*. Sept. 2013, pp. 111–120. DOI: 10.1109/ARES.2013.21.

[Bib77] Ken Biba. *Integrity Considerations for Secure Computer Systems*. Tech. rep. ESD-TR-76-372. MITRE Corporation, Apr. 1977, p. 68.

[BM70] Rudolf Bayer and Edward McCreight. « Organization and Maintenance of Large Ordered Indices ». In: *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*. SIGFIDET '70. Houston, Texas: Association for Computing Machinery, 1970, 107–141. ISBN: 9781450379410. DOI: 10.1145/1734663.1734671. URL: https://doi.org/10.1145/1734663.1734671.

[Bod+09] Stephan Bode, Anja Fischer, Winfried Kühnhauser, and Matthias Riebisch. « Software Architectural Design Meets Security Engineering ». In: *2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*. 2009, pp. 109–118. DOI: 10.1109/ECBS.2009.17.

[Boe87] Barry W Boehm. « Industrial software metrics top 10 list ». In: *IEEE software* 4.5 (1987), pp. 84–85.

[Cas+20] Valentina Casola, Alessandra De Benedictis, Massimiliano Rak, and Umberto Villano. « A novel Security-by-Design methodology: modeling and assessing security with a quantitative approach ». In: *Journal of Systems and Software* 163 (May 2020), p. 110537. DOI: 10.1016/j.jss.2020.110537.

[CD13] Ann Cavoukian and Mark Dixon. *Privacy and security by design: An enterprise architecture approach*. Information and Privacy Commissioner of Ontario, Canada, 2013.

[Cen17] Microsoft Security Response Center. *Guidance for WannaCrypt attacks*. 2017. URL: https://msrc-blog.microsoft.com/2017/05/12/customer-guidance-for-wannacrypt-attacks/.

[Cer+16] Humberto Cervantes, Rick Kazman, Jungwoo Ryoo, Duyoung Choi, and Duksung Jang. « Architectural Approaches to Security: Four Case Studies ». In: *Computer* 49.11 (2016), pp. 60–67. ISSN: 1558-0814. DOI: 10.1109/MC.2016.332.

[Cha+17] A. M. Chamberlain, S. M. Dunlay, Yariv Gerber, S. M. Manemann, R. Jiang, S. A. Weston, and V. L. Roger. « Burden and Timing of Hospitalizations in Heart Failure: A Community Study ». In: *Mayo Clinic proceedings* 92.2 (2017), pp. 184–192. DOI: 10.1016/j.mayocp.2016.11.009. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5341602/.

[CHH16] Golriz Chehrazi, Irina Heimbach, and O. Hinz. « The Impact of Security by Design on the Success of Open Source Software ». In: *ECIS*. 2016.

[Cis] *Cisco Secure Development Lifecycle. (n.d.)*. URL: http://www.cisco.com/web/about/security/cspo/csdl/index.html.

[Coc06] Alistair Cockburn. *Agile Software Development: The Cooperative Game (2nd Edition) (Agile Software Development Series)*. Addison-Wesley Professional, 2006. ISBN: 0321482751.

[Cor18] Microsoft Corporation. *SDL Threat Modeling Tool. Security Development Lifecycle.* 2018. URL: https://www.microsoft.com/en-us/sdl/adopt/threatmodeling.aspx.

[CR18] Eric Coatanéa and Ric Roca. « Dimensional Analysis Conceptual Modeling Supporting Adaptable Reasoning in simulation-based training ». In: *2018 13th Annual Conference on System of Systems Engineering (SoSE)*. 2018, pp. 245–252. DOI: 10.1109/SYSOSE.2018.8428785.

[Cun02] Hamish Cunningham. « GATE, a general architecture for text engineering ». In: *Computers and the Humanities* 36.2 (2002), pp. 223–254.

[Cyb19] Cybint. *15 Alarming Cyber Security Facts and Stats*. Sept. 2019. URL: https://www.cybintsolutions.com/cyber-security-facts-stats/.

[Der+18] Mustapha Derras, L. Deruelle, N. Lévy, and F. Losavio. « Towards Bridging the Gap between Domain and Application Design ». In: *2018 Sixth International Conference on Enterprise Systems (ES)* (2018), pp. 44–49.

[Dhi11] D. Dhillon. « Developer-Driven Threat Modeling: Lessons Learned in the Trenches ». In: *IEEE Security Privacy* 9.4 (2011), pp. 41–47. ISSN: 1558-4046. DOI: 10.1109/MSP.2011.47.

[DoD02] DoD. *Public Key Infrastructure and Key Management Infrastructure Token Protection Profile*. Tech. rep. Common criteria for information technology security evaluation, 2002.

[Don03] Marc Donner. « Toward a Security Ontology ». In: *IEEE Security and Privacy* 1.3 (May 2003), 6–7. ISSN: 1540-7993.

[Dou+09] Chad Dougherty, Kirk Sayre, Robert Seacord, David Svoboda, and Kazuya Togashi. *Secure Design Patterns*. Tech. rep. CMU/SEI-2009-TR-010. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2009. URL: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9115.

[EB11] David Elliott Bell. « Bell–La Padula Model ». In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pp. 74–79. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_811. URL: https://doi.org/10.1007/978-1-4419-5906-5_811.

[Eke+06] Andreas Ekelhart, Stefan Fenz, Markus D. Klemen, and Edgar R. Weippl. « Security Ontology: Simulating Threats to Corporate Assets ». In: *Information Systems Security*. Ed. by Aditya Bagchi and Vijayalakshmi Atluri. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 249–259.

[EM10] Dieter Ernst and Sheri Martin. « The Common Criteria for Information Technology Security Evaluation-Implications for China's Policy on Information Security Standards ». In: *East-West Center Working Papers* 108 (2010).

[Fai+15] Shamal Faily, John Lyle, Ivan, and Andrew Simpson. « Usability and security by design: a case study in research and development ». In: (2015).

[FB13] Eduardo Fernandez-Buglioni. *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. 1st. Wiley Publishing, 2013. ISBN: 1119998948.

[Fou17] The OWASP Foundation. *OWASP Attack List*. 2017. URL: https://owasp.org/www-community/attacks/.

[Fou20] The Linux Foundation. *Report on the 2020 FOSS Contributor Survey*. standard. The Linux Foundation & The Laboratory for Innovation Science at Harvard, 2020, pp. 1–96.

[Fow03] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd ed. Object Technology Series. Boston, MA: Addison-Wesley, 2003. ISBN: 978-0-321-19368-1. URL: https://www.safaribooksonline.com/library/view/uml-distilled-a/0321193687/.

[FS09] Ivan Flechais and Angela Sasse. « Stakeholder involvement, motivation, responsibility, communication: How to design usable security in e-Science ». In: *International Journal of Human-Computer Studies* 67 (Apr. 2009), pp. 281–296. DOI: 10.1016/j.ijhcs.2007.10.002.

[Gam+94] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed. Addison-Wesley Professional, 1994. ISBN: 0201633612. URL: http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1.

[Gan+17] Alexander Ganin, Phuoc Quach, Mahesh Panwar, Zachary Collier, Jeffrey Keisler, and Dayton Marchese. « Multicriteria Decision Framework for Cybersecurity Risk Assessment and Management ». In: *Risk Analysis* 40 (Sept. 2017). DOI: 10.1111/risa.12891.

[Gar+10] Deepak Garg, Jason Franklin, Dilsun Kaynar, and Anupam Datta. « Compositional System Security with Interface-Confined Adversaries ». In: *Electronic Notes in Theoretical Computer Science* 265 (2010). Proceedings of the 26th Conference on the Mathematical Foundations of Programming Semantics (MFPS 2010), pp. 49 –71. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2010.08.005. URL: http://www.sciencedirect.com/science/article/pii/S1571066110000848.

[Gee10] David Geer. « Are Companies Actually Using Secure Development Life Cycles? » In: *Computer* 43 (July 2010), pp. 12 –16. DOI: 10.1109/MC.2010.159.

[Gra+12] Tim Grance, Joan Hash, Jonathan Smith, and Karen Korow-Diks. *Security Guide for Interconnecting Information Technology Systems: Recommendations of the National Institute of Standards and Technology.* Tech. rep. SP 800-47. NIST, 2012.

[Gri+18] Dimitris Gritzalis, Giulia Iseppi, Alexios Mylonas, and Vasilis Stavrou. « Exiting the Risk Assessment Maze: A Meta-Survey ». In: *ACM Comput. Surv.* 51.1 (Jan. 2018), 11:1–11:30. ISSN: 0360-0300. DOI: 10.1145/3145905. URL: http://doi.acm.org/10.1145/3145905.

[GSY04] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. « S-Match: an Algorithm and an Implementation of Semantic Matching ». In: *The Semantic Web: Research and Applications.* Ed. by Christoph J. Bussler, John Davies, Dieter Fensel, and Rudi Studer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 61–75. ISBN: 978-3-540-25956-5.

[Hac+20] Jamal EL Hachem, Vanea Chiprianov, Muhammad Ali Babar, Tarek AL Khalil, and Philippe Aniorte. « Modeling, analyzing and predicting security cascading attacks in smart buildings systems-of-systems ». In: *Journal of Systems and Software* 162 (2020), p. 110484. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2019.110484. URL: http://www.sciencedirect.com/science/article/pii/S0164121219302584.

[Han17] Stefan Hanenberg. « Empirical, Human-Centered Evaluation of Programming and Programming Language Constructs: Controlled Experiments ». In: *Grand Timely Topics in Software Engineering*. Ed. by Jácome Cunha, João P. Fernandes, Ralf Lämmel, João Saraiva, and Vadim Zaytsev. Cham: Springer International Publishing, 2017, pp. 45–72. ISBN: 978-3-319-60074-1.

[Has18] Wilhelm Hasselbring. « Software Architecture: Past, Present, Future ». In: *The Essence of Software Engineering*. Ed. by Volker Gruhn and Rüdiger Striemer. Cham: Springer International Publishing, 2018, pp. 169–184. ISBN: 978-3-319-73897-0. DOI: 10.1007/978-3-319-73897-0_10. URL: https://doi.org/10.1007/978-3-319-73897-0_10.

[HL06] Michael Howard and Steve Lipner. *The Security Development Lifecycle*. Redmond, WA, USA: Microsoft Press, 2006. ISBN: 0735622140.

[IEE90] IEEE. *IEEE Standard Glossary of Software Engineering Terminology*. standard IEEE Std 610.12-1990. Institute of Electrical and Electronics Engineers, 1990, pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064.

[ISO01] ISO. *Software engineering – Product quality*. standard ISO/IEC 9126. International Organization for Standardization, 2001. URL: https://www.iso.org/standard/22749.html.

[ISO08] ISO/IEC 21827:2008. *Information technology – Security techniques – Systems Security Engineering – Capability Maturity Model*. Standard ISO/IEC 21827:2008. Geneva, CH: International Organization for Standardization, 2008. URL: https://www.iso.org/standard/44716.html.

[ISO11] ISO. *Information technology - Security techniques - Information security risk management*. standard ISO/IEC 27005. International Organization for Standardization, 2011. URL: https://www.iso.org/standard/56742.html.

[ISO13a] ISO. *Information technology — Security techniques — Code of practice for information security controls*. standard ISO/IEC 27002:2013. International Organization for Standardization, 2013. URL: https://www.iso.org/standard/54533.html.

[ISO13b] ISO. *Information technology — Security techniques — Information security management systems — Requirements*. standard ISO/IEC 27001:2013. 2013. URL: https://www.iso.org/standard/54534.html.

[ISO18] ISO. *Information technology - Security techniques - Information security management systems - Overview and vocabulary*. standard ISO/IEC 27000:2018. 2018. URL: https://www.iso.org/standard/73906.html.

[(IT08] International Telecommunications Union (ITU). *Series X: data networks, open system communications and security: telecommunication security: overview of cybersecurity*. Tech. rep. ITU-T X.1205. 2008. URL: https://www.itu.int/rec/T-REC-X.1205-200804-I.

[JGJ97] Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse: Architecture, Process and Organization for Business Success*. USA: ACM Press/Addison-Wesley Publishing Co., 1997. ISBN: 0201924765.

[JH12] R. A. Jones and B. Horowitz. « A System-Aware Cyber Security Architecture ». In: *System Engineering* 15.2 (June 2012), pp. 225–240. ISSN: 1098-1241. DOI: 10.1002/sys.21206. URL: http://dx.doi.org/10.1002/sys.21206.

[Joh+16] Pontus Johnson, Alexandre Vernotte, Mathias Ekstedt, and Robert Lagerström. « pwnPr3d: An Attack-Graph-Driven Probabilistic Threat-Modeling Approach. » In: *Proceedings of International Conference on Availability, Reliability and Security (ARES)*. IEEE Computer Society, 2016, pp. 278–283. ISBN: 978-1-5090-0990-9. URL: http://dblp.uni-trier.de/db/conf/IEEEares/ares2016.html#JohnsonVEL16.

[JR20] Abbas Javan Jafari and Abbas Rasoolzadegan. « Security Patterns: A Systematic Mapping Study ». In: *Journal of Computer Languages* 56 (2020).

[Jür02] Jan Jürjens. « UMLsec: Extending UML for Secure Systems Development ». In: *UML 2002 — The Unified Modeling Language*. Ed. by Jean-Marc Jézéquel, Heinrich Hussmann, and Stephen Cook. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 412–425. ISBN: 978-3-540-45800-5.

[Kam16] Kamatchi, R. and Ambekar, Kimaya. « Analyzing Impacts of Cloud Computing Threats in Attack based Classification Models ». In: *Indian journal of science and technology* 9.21 (2016).

[KG99] Loren Kohnfelder and Praerit Garg. « The threats to our products ». In: *Microsoft Interface, Microsoft Corporation* 33 (1999).

[Kis+08] Richard Kissel, Kevin M Stine, Matthew A Scholl, Hart Rossman, James Fahlsing, and Jessica Gulick. *Security considerations in the system development life cycle, revision 2*. Tech. rep. SP 800-64. NIST, 2008.

[KJ07] Michael Kenward and Byron Jones. « The Design and Analysis of Cross-Over Trials ». In: *Handbook of Statistics* 27 (Dec. 2007), pp. 464–490. DOI: 10.1016/S0169-7161(07)27015-4.

[Kum+18] Rajesh Kumar, Stefano Schivo, Enno Ruijters, Buğra Mehmet Yildiz, David Huistra, Jacco Brandt, Arend Rensink, and Mariëlle Stoelinga. « Effective Analysis of Attack Trees: A Model-Driven Approach ». In: *Fundamental Approaches to Software Engineering*. Ed. by Alessandra Russo and Andy Schürr. Cham: Springer, 2018, pp. 56–73. ISBN: 978-3-319-89363-1.

[LBD02] Torsten Lodderstedt, David Basin, and Jürgen Doser. « SecureUML: A UML-Based Modeling Language for Model-Driven Security ». In: *UML 2002 — The Unified Modeling Language*. Ed. by Jean-Marc Jézéquel, Heinrich Hussmann, and Stephen Cook. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 426–441. ISBN: 978-3-540-45800-5.

[Lee13a] Ruby Lee. « Improving Cyber Security ». In: *Advances in Cyber Security: Technology, Operations, and Experiences*. Ed. by Frank D. Hsu and Dorothy Marinucci. Fordham Scholarship Online, 2013. DOI: 10.5422/fordham/9780823244560.003.0001.

[Lee13b] Ruby B. Lee. *Security Basics for Computer Architects*. Morgan amp; Claypool Publishers, 2013. ISBN: 1627051554.

[Mal+19] Sam Malek, Hamid Bagheri, Joshua Garcia, and Alireza Sadeghi. « Security and Software Engineering ». In: *Handbook of Software Engineering*. Ed. by Sungdeok Cha, Richard N. Taylor, and Kyochul Kang. Cham: Springer International Publishing, 2019, pp. 445–489. ISBN: 978-3-030-00262-6. DOI: 10.1007/978-3-030-00262-6_12. URL: https://doi.org/10.1007/978-3-030-00262-6_12.

[McB07] Matthew McBride. « The software architect ». In: *Communications of the ACM* 50.5 (May 2007), pp. 75–81. DOI: 10.1145/1230819.1241667.

[McG06] Gary McGraw. *Software Security: Building Security In*. Addison-Wesley Professional, 2006. ISBN: 0321356705.

[MD10] Raimundas Matulevicius and Marlon Dumas. « A Comparison of SecureUML and UMLsec for Role-based Access Control ». In: *Proceedings of the 9th Conference on Databases and Information Systems*. Riga, Latvia, 2010, pp. 171–185.

[MEL01] Andrew Moore, Robert Ellison, and Rick Linger. *Attack Modeling for Information Security and Survivability*. Tech. rep. CMU/SEI-2001-TN-001. Software Engineering Institute, Carnegie Mellon Institute, 2001.

[Mel+10] Daniel Mellado, Carlos Blanco, Luis E. Sánchez, and Eduardo Fernández-Medina. « A systematic review of security requirements engineering ». In: *Computer Standards Interfaces* 32.4 (2010), pp. 153 –165. ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2010.01.006. URL: http://www.sciencedirect.com/science/article/pii/S0920548910000255.

[Mes+19] Nan Messe, Nicolas Belloir, Vanea Chiprianov, Imane Cherfa, Régis Fleurquin, and Salah Sadou. « Development of Secure System of Systems Needing a Rapid Deployment ». In: *2019 14th Annual Conference System of Systems Engineering (SoSE)*. 2019. DOI: 10.1109/SYSOSE.2019.8753857.

[MFMP07] Daniel Mellado, Eduardo Fernández-Medina, and Mario Piattini. « A Common Criteria Based Security Requirements Engineering Process for the Development of Secure Information Systems ». In: *Comput. Stand. Interfaces* 29.2 (Feb. 2007), 244–253. ISSN: 0920-5489. DOI: 10.1016/j.csi.2006.04.002. URL: https://doi.org/10.1016/j.csi.2006.04.002.

[MITa] MITRE. *ATT&CK Matrix for Enterprise*. URL: https://attack.mitre.org/.

[MITb] MITRE. *Common Attack Pattern Enumeration and Classification*. URL: http://capec.mitre.org/.

[MITc] MITRE. *Common Platform Enumeration. https://cpe.mitre.org/*. URL: https://cpe.mitre.org/.

[MITd] MITRE. *Common Vulnerabilities and Exposures*. URL: https://cve.mitre.org/.

[MITe] MITRE. *Common Weakness Enumeration. https://cwe.mitre.org*. URL: https://cwe.mitre.org/cwraf/enum_of_ti.html.

[MKM15] Zhendong Ma, Friederich Kupzog, and Paul Murdock. « Secure Development Life Cycle ». In: *Smart Grid Security*. Elsevier, 2015, pp. 219–245.

[MO06] Sjouke Mauw and Martijn Oostdijk. « Foundations of Attack Trees ». In: *Proceedings of the 8th International Conference on Information Security and Cryptology*. LNCS. Seoul, Korea: Springer-Verlag, 2006, pp. 186–198. ISBN: 978-3-540-33354-8. DOI: 10.1007/11734727_17. URL: http://dx.doi.org/10.1007/11734727_17.

[Moh+17] Nabil M. Mohammed, Mahmood Niazi, Mohammad Alshayeb, and Sajjad Mahmood. « Exploring software security approaches in software development lifecycle: A systematic mapping study ». In: *Computer Standards Interfaces* 50 (2017), pp. 107 –115. ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2016.10.001. URL: http://www.sciencedirect.com/science/article/pii/S0920548916301155.

[MSR07] Peter Mell, Karen Scarfone, and Sasha Romanosky. *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. 1st ed. NIST and Carnegie Mellon University. 2007. URL: http://www.first.org/cvss/cvss-guide.html.

[MT18] Kilisio Mercy and Moses Thiga. « Practices, Challenges and Approaches for Software Project Risk Management in Kenyan County Governments ». In: *8th Annual Conference Kabarak University*. 2018.

[Ngu+15] Phu Nguyen, Max Kramer, Jacques Klein, and Yves Le Traon. « An Extensive Systematic Review on the Model-Driven Development of Secure Systems ». In: *Information and Software Technology* 68 (Sept. 2015), 62–81. DOI: 10.1016/j.infsof.2015.08.006.

[NIS12] NIST. *Guide for Conducting Risk Assessments 2012*. Tech. rep. SP 800-30. National Institute of Standards and Technology, 2012. URL: https://csrc.nist.gov/publications/detail/sp/800-30/rev-1/final.

[NIS13] NIST. *Recommended Security and Privacy Controls for Federal Information 1235 Systems and Organizations (Revision 4)*. Tech. rep. SP 800-53. NIST, 2013.

[OMG15] OMG. *OMG Unified Modeling Language (UML), Version 1.6*. Object Management Group, 2015. URL: https://www.omg.org/spec/UML/2.5/About-UML/.

[OMG19] OMG. *OMG Systems Modeling Language (SysML), Version 1.6*. Object Management Group, 2019. URL: https://www.omg.org/spec/SysML/1.6/.

[OSC06] Ebenezer A. Oladimeji, Sam Supakkul, and Lawrence Chung. « Security threat modeling and analysis: A goal-oriented approach ». In: *ICSE 2006*. 2006.

[PBL05] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Berlin: Springer, 2005. ISBN: 978-3-540-24372-4. DOI: 10.1007/3-540-28901-1.

[PC99] Mary Putt and Vernon M. Chinchilli. « A mixed effects model for the analysis of repeated measures cross-over studies ». In: *Statistics in Medicine* 18.22 (Nov. 1999), pp. 3037–3058. ISSN: 0277-6715. DOI: 10.1002/(SICI)1097-0258(19991130)18:22<3037::AID-SIM243>3.0.CO;2-7.

[Pet77] James L Peterson. « Petri Nets ». In: *ACM Comput. Surv.* 9.3 (Sept. 1977), pp. 223–252. ISSN: 0360-0300. DOI: 10.1145/356698.356702. URL: http://doi.acm.org/10.1145/356698.356702.

[Pre94] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. European. Adapted by Darrel Ince. McGraw-Hill, 1994.

[Rau+16] Tobias Rauter, Andrea Höller, Johannes Iber, and Christian Kreiner. « Asset-Centric Security Risk Assessment of Software Components. » In: *Workshop on MILS: Architecture and Assurance for Secure Systems*. Jan. 2016.

[Res17] Symantec Security Response. *Petya ransomware: Here's what you need to know*. June 2017. URL: https://www.symantec.com/blogs/threat-intelligence/petya-ransomware-wiper.

[Rhe+13] Keunwoo Rhee, Dongho Won, Sang-Woon Jang, Sooyoung Chae, and Sangwoo Park. « Threat modeling of a mobile device management system for secure smart work ». In: *Electronic Commerce Research* 13 (Sept. 2013). DOI: 10.1007/s10660-013-9121-4.

[RMO16] Ron S. Ross, Michael McEvilley, and Janet c. Oren. *Systems security engineering: Considerations for a multidisciplinary approach in the engineering of trustworthy secure systems*. Tech. rep. SP 800-160. NIST, 2016.

[San+17] Joanna CS Santos, Anthony Peruma, Mehdi Mirakhorli, Matthias Galstery, Jairo Veloz Vidal, and Adriana Sejfia. « Understanding Software Vulnerabilities Related to Architectural Security Tactics: An Empirical Investigation of Chromium, PHP and Thunderbird ». In: *2017 IEEE International Conference on Software Architecture (ICSA)*. 2017, pp. 69–78. DOI: 10.1109/ICSA.2017.39.

[SB14] William Stallings and Lawrie Brown. *Computer Security: Principles and Practice*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014. ISBN: 0133773922, 9780133773927.

[Sch+09] Daniel Scherr, Peter Kastner, Alexander Kollmann, Andreas Hallas, Johann Auer, Heinz Krappinger, Herwig Schuchlenz, Gerhard Stark, Wilhelm Grander, Gabriele Jakl, et al. « Effect of Home-Based Telemonitoring Using Mobile Phone Technology on the Outcome of Heart Failure Patients After an Episode of Acute Decompensation: Randomized Controlled Trial ». In: *Journal of Medical Internet Research* 11.3 (2009), e34. ISSN: 1438-8871. DOI: 10.2196/jmir.1252. URL: http://www.ncbi.nlm.nih.gov/pubmed/19687005.

[Sch99] Bruce Schneier. « Attack trees ». In: *Dr. Dobb's Journal* (Dec. 1999).

[SDP08] Vineet Saini, Qiang Duan, and Vamsi Paruchuri. « Threat Modeling Using Attack Trees ». In: *Journal of Computing Sciences in Colleges* 23 (Apr. 2008).

[She+18] Nataliya Shevchenko, Timothy A Chick, Paige O'Riordan, Thomas P Scanlon, and Carol Woody. *Threat Modeling: a Summary of Available Methods*. White paper. Software Engineering Institute, Carnegie Mellon University, 2018.

[Sho08] Adam Shostack. « Experiences Threat Modeling at Microsoft ». In: *Proceedings of the MODSEC@MoDELS workhop*. 2008.

[Sho14] Adam Shostack. *Threat Modeling: Designing for Security*. 1st. Wiley Publishing, 2014. ISBN: 9781118809990.

[SHS09] Amril Syalim, Yoshiaki Hori, and Kouichi Sakurai. « Comparison of Risk Analysis Methods: Mehari, Magerit, NIST800-30 and Microsoft's Security Management Guide ». In: *Proceedings of the International Conference on Availability, Reliability and Security*. 2009, pp. 726–731. DOI: 10.1109/ARES.2009.75.

[Sim14] Stacy Simpson. « SAFECode Whitepaper: Fundamental Practices for Secure Software Development 2nd Edition. » In: *ISSE*. Ed. by Helmut Reimer, Norbert Pohlmann, and Wolfgang Schneider. Springer, 2014, pp. 1–32. ISBN: 978-3-658-06708-3. DOI: 10.1007/978-3-658-06708-3. URL: http://dblp.uni-trier.de/db/conf/isse/isse2014.html#Simpson14.

[SL17] Gianluigi Savarese and Lars H Lund. « Global Public Health Burden of Heart Failure ». In: *Cardiac failure review* 3.1 (2017), pp. 7–11. DOI: 10.15420/cfr.2016:25:2. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5494150/.

[SO03] Guttorm Sindre and Andreas Opdahl. « A Reuse-Based Approach to Determining Security Requirements ». In: *Proceedings of the 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ03)*. May 2003.

[Som10] Ian Sommerville. *Software Engineering*. 9th. USA: Addison-Wesley Publishing Company, 2010. ISBN: 0137035152.

[Ste10] John Steven. « Threat Modeling - Perhaps It's Time ». In: *IEEE Security Privacy* 8.3 (2010), pp. 83–86. ISSN: 1558-4046. DOI: 10.1109/MSP.2010.110.

[SWJ13] Riccardo Scandariato, Kim Wuyts, and Wouter Joosen. « A descriptive study of Microsoft's threat modeling technique ». In: *Requirements Engineering* 20 (June 2013). DOI: 10.1007/s00766-013-0195-2.

[TBB18] Giannis Tziakouris, Rami Bahsoon, and Muhammad Ali Babar. « A Survey on Self-Adaptive Security for Large-scale Open Environments ». In: *ACM Comput. Surv.* 51.5 (Oct. 2018), 100:1–100:42. ISSN: 0360-0300. DOI: 10.1145/3234148. URL: http://doi.acm.org/10.1145/3234148.

[TÇS18] Katja Tuma, Gül Çalikli, and Riccardo Scandariato. « Threat analysis of software systems: A systematic literature review ». In: *Journal of Systems and Software* 144 (2018), pp. 275 –294. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2018.06.073. URL: http://www.sciencedirect.com/science/article/pii/S0164121218301304.

[Tec19] TechTarget. *BlueKeep (CVE-2019-0708)*. June 2019. URL: https://searchsecurity.techtarget.com/definition/BlueKeep-CVE-2019-0708.

[TJR10] Inger Anne Tøndel, Jostein Jensen, and Lillian Røstad. « Combining Misuse Cases with Attack Trees and Security Activity Models ». In: *2010 International Conference on Availability, Reliability and Security*. 2010, pp. 438–445. DOI: 10.1109/ARES.2010.101.

[TNP15] Quoc-Cuong To, Benjamin Nguyen, and Philippe Pucheral. « TrustedMR: A Trusted MapReduce System Based on Tamper Resistance Hardware ». In: *On the Move to Meaningful Internet Systems: OTM 2015 Conferences*. Ed. by Christophe Debruyne, Hervé Panetto, Robert Meersman, Tharam Dillon, Georg Weichhart, Yuan An, and Claudio Agostino Ardagna. Cham: Springer International Publishing, 2015, pp. 38–56. ISBN: 978-3-319-26148-5.

[Tor05] Peter Torr. « Demystifying the threat modeling process ». In: *IEEE Security Privacy* 3.5 (2005), pp. 66–70. ISSN: 1558-4046. DOI: 10.1109/MSP.2005.119.

[Tum+19] Katja Tuma, Danial Hosseini, Kyriakos Malamas, and Riccardo Scandariato. « Inspection Guidelines to Identify Security Design Flaws ». In: *Proceedings of the 13th European Conference on Software Architecture - Volume 2*. ECSA '19. Paris, France: ACM, 2019, pp. 116–122. ISBN: 978-1-4503-7142-1. DOI: 10.1145/3344948.3344995. URL: http://doi.acm.org/10.1145/3344948.3344995.

[Tzi+16] Giannis Tziakouris, Marios Zinonos, Tom Chothia, and Rami Bahsoon. « Asset-centric Security-Aware Service Selection ». In: *2016 IEEE International Congress on Big Data (BigData Congress)*. 2016, pp. 327–332. DOI: 10.1109/BigDataCongress.2016.50.

[Uce12] Tony UcedaVelez. « Real world threat modeling using the pasta methodology ». In: *OWASP App Sec EU* (2012).

[UF14] Anton V. Uzunov and Eduardo B. Fernandez. « An extensible pattern-based library and taxonomy of security threats for distributed systems ». In: *Computer Standards Interfaces* 36.4 (2014). Security in Information Systems: Advances and new Challenges, pp. 734 –747. ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2013.12.008. URL: http://www.sciencedirect.com/science/article/pii/S0920548913001827.

[VAJ15] Sira Vegas, Cecilia Apa, and Natalia Juristo. « Cross-Over Designs in Software Engineering Experiments: Benefits and Perils ». In: *IEEE Transactions on Software Engineering* 42 (Jan. 2015), pp. 1–1. DOI: 10.1109/TSE.2015.2467378.

[Var20] Varonis. *Analyzing Company Reputation After a Data Breach*. Tech. rep. 2020. URL: https://www.varonis.com/blog/company-reputation-after-a-data-breach/.

[VDB+17] Alexander Van Den Berghe, Riccardo Scandariato, Koen Yskout, and Wouter Joosen. « Design Notations for Secure Software: A Systematic Literature Review ». In: *Software and Systems Modeling* 16.3 (2017), pp. 809–831. ISSN: 1619-1366. DOI: 10.1007/s10270-015-0486-9.

[VDB+18] A. Van Den Berghe, K. Yskout, R. Scandariato, and W. Joosen. « A lingua franca for security by design ». In: *3rd Annual IEEE Cybersecurity Development Conference (IEEE SecDev)*. IEEE COMPUTER SOC, Nov. 2018, pp. 69–76. ISBN: 9781538676622. DOI: https://doi.org/10.1109/SecDev.2018.00017. URL: https://lirias.kuleuven.be/2359992.

[Vli07] Hans van Vliet. *Software engineering - principles and practice*. Willey, 2007. ISBN: 978-0470031469.

[VSVN13] Rossouw Von Solms and Johan Van Niekerk. « From Information Security to Cyber Security ». In: *Comput. Secur.* 38 (Oct. 2013), 97–102. ISSN: 0167-4048. DOI: 10.1016/j.cose.2013.04.004. URL: https://doi.org/10.1016/j.cose.2013.04.004.

[XR19] Wenjun Xiong and Lagerström Robert. « Threat Modeling – A Systematic Literature Review ». In: *Computers Security* 84 (Mar. 2019), pp. 53–69. DOI: 10.1016/j.cose.2019.03.010.

[YB97] Joseph Yoder and Jeffrey Barcalow. « Architectural patterns for enabling application security ». In: *Proceedings of the 4th Conference on Patterns Language of Programming (PLoP'97)*. Vol. 2. Citeseer. 1997.

[Ysk+20] Koen Yskout, Thomas Heyman, Dimitri Van Landuyt, Laurens Sion, Kim Wuyts, and Wouter Joosen. « Threat modeling: from infancy to maturity ». In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*. 2020, pp. 9–12.

[Zho12] Lei Zhou. « Application of linear mixed-effects models to crossover designs ». MA thesis. Louisville, Kentucky, USA: University of Louisville, 2012.

Part V

Appendices

Appendix A

Results of period 1 with group 1 by applying the Microsoft SDL tool

Subject 1:

1. Potential SQL Injection Vulnerability for SQL Database (Category: Tampering), appeared two times on the two Generic Data Flows between OS Process and SQL Database;
2. Spoofing of Destination Data Store SQL Database (Category: Spoofing), appeared four times on the two Generic Data Flows between OS Process and SQL Database;
3. Potential Excessive Resource Consumption for OS Process or SQL Database (Category: Denial of Service), appeared two times on the two Generic Data Flows between OS Process and SQL Database;
4. Weak Access Control for a Resource (Category: Information Disclosure), appeared three times on the SQL Database.

Subject 5 and subject 4:

1. Spoofing of Destination Data Store (Category: Spoofing), appeared four times on the Data Store Device - ICD and Database Server;
2. Data Logs from an Unknown Source (Category: Repudiation), appeared three times;
3. Lower Trusted Subject Updates Logs (Category: Repudiation), appeared three times;
4. Possible SQL Injection Vulnerability for Database server (Category: Tampering), appeared three times on the Database Server;
5. Weak Access Control for a Resource (Category: Information Disclosure), appeared two times on the Device -ICD;

6. Spoofing of Source Data Store Device - ICD (Category: Spoofing), appeared two times on the Device - ICD;
7. Risks from Logging (Category: Tampering), appeared two times;
8. Insufficient Auditing (Category: Repudiation), appeared two times;
9. Potential Weak Protections for Audit Data (Category: Repudiation), appeared two times;
10. Weak Credential Storage (Category: Information Disclosure), appeared two times;
11. Potential Excessive Resource Consumption for Processing server or Database server (Category: Denial of Service), appeared two times on Database Server;
12. Spoofing the Navigateur internet External Entity (Category: Spoofing), appeared two times on Browser;
13. Cross Site Scripting (Category: Tampeing), appeared three times on Processing Server and Monitoring Server;
14. Persistent Cross Site Scripting (Category: Tampering), appeared one time on Monitoring Server;
15. Elevation Using Impersonation (Category: Elevation of Privilege), appeared two times on Browser.

Appendix B

Results of period 1 with group 2 by applying the security assistance tool

Subject 2 & subject 3:

1. Standard Attack Pattern:
   - SQL Injection (CAPEC-66), for the VDA "Aerospike database server's untested SQL input";
2. Detailed Attack Pattern:
   - Read Sensitive Constants Within an Executable (CAPEC-191), for the VDA "Medtronic monitor's hard coded credential" and the VDA "Medtronic monitor's readable sensitive information";
   - Retrieve Embedded Sensitive Data (CAPEC-37), for the VDA "Medtronic monitor's readable sensitive information";
   - Lifting Sensitive Data Embedded in Cache (CAPEC-204), for the VDA "Medtronic monitor's readable sensitive information";
   - Utilizing REST's Trust in the System Resource to Obtain Sensitive Data (CAPEC-57), for the VDA "Medtronic monitor's readable sensitive information";
   - Fuzzing for garnering other adjacent user/sensitive data (CAPEC-261), for the VDA "Medtronic monitor's readable sensitive information";
   - Screen Temporary Files for Sensitive Information (CAPEC-155), for the VDA "Medtronic monitor's readable sensitive information";
   - Blind SQL Injection (CAPEC-7), for the VDA "Aerospike database server's untested SQL input";

— SQL Injection through SOAP Parameter Tampering (CAPEC-110), for the VDA “Aerospike database server’s untested SQL input”;

— Command Line Execution through SQL Injection (CAPEC-108), for the VDA “Aerospike database server’s untested SQL input”.

Subject 6 and subject 7:

1. Meta Attack Pattern:

   — Reverse Engineering (CAPEC-188), for the VDA “Medtronic 24950 MaCare-Link’s software function”.

2. Standard Attack Pattern:

   — White Box Reverse Engineering (CAPEC-167), for the VDA “Medtronic 24950 MaCareLink’s executable”.

   — SQL Injection (CAPEC-66), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”;

   — HTTP Flood (CAPEC-488), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”;

   — UDP Flood (CAPEC-486), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”;

   — UDP Fragmentation (CAPEC-495), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”;

   — HTTP DoS (CAPEC-469), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”;

   — XML Injection (CAPEC-250), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”;

   — Web Services Protocol Manipulation (CAPEC-278), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”.

3. Detailed Attack Pattern:

   — HTTP Request Smuggling (CAPEC-33), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”;

   — HTTP Response Splitting (CAPEC-34), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”;

— HTTP Response Smuggling (CAPEC-273), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”;

— Blind SQL Injection (CAPEC-7), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”;

— SQL Injection through SOAP Parameter Tampering (CAPEC-110), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”;

— Command Line Execution through SQL Injection (CAPEC-108), for the VDA “Aerospike database server 3.10.0.3’s untested SQL input”;

— Read Sensitive Constants Within an Executable (CAPEC-191), for the VDA “Medtronic monitor’s hard coded credential” and the VDA “Medtronic monitor’s readable sensitive information”.

Appendix C

Results of period 2 with group 1 by applying the security assistance tool

Subject 1:

1. Meta Attack Pattern:
   - Authentication Bypass (CAPEC-115), for the VDA "Authentication mechanism's manipulable authentication protocol";
   - Authentication Abuse (CAPEC-114), for the VDA "Authentication mechanism's manipulable authentication protocol";
2. Standard Attack Pattern:
   - XML Injection (CAPEC-250), for the VDA "Web service's untested XML input" and "Database server's untested XML input";
   - Counterfeit GPS Signals (CAPEC-627), for the VDA "GPS's receiver's modifiable GPS signal";
   - Web Services Protocol Manipulation (CAPEC-278), for the VDA "HTTP server";
   - HTTP Request Splitting (CAPEC-105), for the VDA "HTTP server";
   - HTTP DoS (CAPEC-469), for the VDA "HTTP server";
   - HTTP Flood (CAPEC-488), for the VDA "HTTP server";
   - UDP Fragmentation (CAPEC-495), for the VDA "UDP server";
   - UDP Flood (CAPEC-486), for the VDA "UDP server";
   - Reflection Attack in Authentication Protocol (CAPEC-90), for the VDA "Authentication mechanism's manipulable authentication protocol";
3. Detailed Attack Pattern:

— Carry-Off GPS Attack (CAPEC-628), for the VDA “GPS’s receiver’s modifiable GPS signal”;

— Cloning RFID Cards or Chips (CAPEC-399), for the VDA “RFID system’s modifiable RFID tag”;

— RFID Chip Deactivation or Destruction (CAPEC-400), for the VDA “RFID system’s modifiable RFID tag”;

— HTTP Response Splitting (CAPEC-34), for the VDA “HTTP server”;

— HTTP Request Smuggling (CAPEC-33), for the VDA “HTTP server”;

— HTTP Verb Tampering (CAPEC-274), for the VDA “HTTP server”;

— HTTP Response Smuggling (CAPEC-273), for the VDA “HTTP server”;

— Retrieve Embedded Sensitive Data (CAPEC-37), for the VDA “Database server’s readable sensitive information”;

— Lifting Sensitive Data Embedded in Cache (CAPEC-204), for the VDA “Database server’s readable sensitive information”;

— Utilizing REST’s Trust in the System Resource to Obtain Sensitive Data (CAPEC-57), for the VDA “Database server’s readable sensitive information”;

— Fuzzing for garnering other adjacent user/sensitive data (CAPEC-261), for the VDA “Database server’s readable sensitive information”;

— Screen Temporary Files for Sensitive Information (CAPEC-155), for the VDA “Database server’s readable sensitive information”;

— Read Sensitive Constants Within an Executable (CAPEC-191), for the VDA “Database server’s readable sensitive information”.

Subject 4:

1. Meta Attack Pattern:

   — Authentication Bypass (CAPEC-115), for the VDA “Authentication mechanism’s manipulable authentication protocol”;

   — Authentication Abuse (CAPEC-114), for the VDA “Authentication mechanism’s manipulable authentication protocol”.

2. Standard Attack Pattern:

   — XML Injection (CAPEC-250), for the VDA “Web service’s untested XML input” and “Database server’s untested XML input”;

— Counterfeit GPS Signals (CAPEC-627), for the VDA “GPS’s receiver’s modifiable GPS signal”;

— Web Services Protocol Manipulation (CAPEC-278), for the VDA “HTTP server”;

— HTTP Request Splitting (CAPEC-105), for the VDA “HTTP server”;

— HTTP DoS (CAPEC-469), for the VDA “HTTP server”;

— Reflection Attack in Authentication Protocol (CAPEC-90), for the VDA “Authentication mechanism’s manipulable authentication protocol”.

3. Detailed Attack Pattern:

— Cloning RFID Cards or Chips (CAPEC-399), for the VDA “RFID system’s modifiable RFID tag”;

— RFID Chip Deactivation or Destruction (CAPEC-400), for the VDA “RFID system’s modifiable RFID tag”;

— Carry-Off GPS Attack (CAPEC-628), for the VDA “GPS’s receiver’s modifiable GPS signal”;

— Read Sensitive Constants Within an Executable (CAPEC-191), for the VDA “Database server’s readable sensitive information”;

— Screen Temporary Files for Sensitive Information (CAPEC-155), for the VDA “Database server’s readable sensitive information”;

— Fuzzing for garnering other adjacent user/sensitive data (CAPEC-261), for the VDA “Database server’s readable sensitive information”;

— Utilizing REST’s Trust in the System Resource to Obtain Sensitive Data (CAPEC-57), for the VDA “Database server’s readable sensitive information”;

— Lifting Sensitive Data Embedded in Cache (CAPEC-204), for the VDA “Database server’s readable sensitive information”;

— Retrieve Embedded Sensitive Data (CAPEC-37), for the VDA “Database server’s readable sensitive information”;

— HTTP Verb Tampering (CAPEC-274), for the VDA “HTTP server”;

— HTTP Response Smuggling (CAPEC-273), for the VDA “HTTP server”;

— HTTP Response Splitting (CAPEC-34), for the VDA “HTTP server”;

— HTTP Request Smuggling (CAPEC-33), for the VDA “HTTP server”.

Subject 5:

1. Meta Attack Pattern:
   - Authentication Bypass (CAPEC-115), for the VDA “Authentication mechanism’s manipulable authentication protocol”;
   - Authentication Abuse (CAPEC-114), for the VDA “Authentication mechanism’s manipulable authentication protocol”.
2. Standard Attack Pattern:
   - XML Injection (CAPEC-250), for the VDA “Web service’s untested XML input” and “Database server’s untested XML input”;
   - SQL Injection (CAPEC-66), for the VDA “Database server’s untested SQL input”;
   - Counterfeit GPS Signals (CAPEC-627), for the VDA “GPS’s receiver’s modifiable GPS signal”;
   - Web Services Protocol Manipulation (CAPEC-278), for the VDA “HTTP server”;
   - HTTP Request Splitting (CAPEC-105), for the VDA “HTTP server”;
   - HTTP DoS (CAPEC-469), for the VDA “HTTP server”;
   - HTTP Flood (CAPEC-488), for the VDA “HTTP server”;
   - UDP Fragmentation (CAPEC-495), for the VDA “UDP server”;
   - UDP Flood (CAPEC-486), for the VDA “UDP server”;
   - Reflection Attack in Authentication Protocol (CAPEC-90), for the VDA “Authentication mechanism’s manipulable authentication protocol”.
3. Detailed Attack Pattern:
   - Command Line Execution through SQL Injection (CAPEC-108), for the VDA “Database server’s untested SQL input”;
   - SQL Injection through SOAP Parameter Tampering (CAPEC-110), for the VDA “Database server’s untested SQL input”;
   - Blind SQL Injection (CAPEC-7), for the VDA “Database server’s untested SQL input”;
   - Carry-Off GPS Attack (CAPEC-628), for the VDA “GPS’s receiver’s modifiable GPS signal”;

— Cloning RFID Cards or Chips (CAPEC-399), for the VDA “RFID system’s modifiable RFID tag”;

— RFID Chip Deactivation or Destruction (CAPEC-400), for the VDA “RFID system’s modifiable RFID tag”;

— HTTP Response Splitting (CAPEC-34), for the VDA “HTTP server”;

— HTTP Request Smuggling (CAPEC-33), for the VDA “HTTP server”;

— HTTP Verb Tampering (CAPEC-274), for the VDA “HTTP server”;

— HTTP Response Smuggling (CAPEC-273), for the VDA “HTTP server”;

— Retrieve Embedded Sensitive Data (CAPEC-37), for the VDA “Database server’s readable sensitive information”;

— Lifting Sensitive Data Embedded in Cache (CAPEC-204), for the VDA “Database server’s readable sensitive information”;

— Utilizing REST’s Trust in the System Resource to Obtain Sensitive Data (CAPEC-57), for the VDA “Database server’s readable sensitive information”;

— Fuzzing for garnering other adjacent user/sensitive data (CAPEC-261), for the VDA “Database server’s readable sensitive information”;

— Screen Temporary Files for Sensitive Information (CAPEC-155), for the VDA “Database server’s readable sensitive information”;

— Read Sensitive Constants Within an Executable (CAPEC-191), for the VDA “Database server’s readable sensitive information”.

Appendix D

Results of period 2 with group 2 by applying the Microsoft SDL tool

Subjects 2 & subject 3:

1. Cross Site Scripting (Category: Tampering), appeared four times on the Web Server and the Web application;
2. Persistent Cross Site Scripting (Category: Tampering), on the HTTPS data flow between Web Server and SQL Database;
3. Weak Access Control for a Resource (Category: Information Disclosure), on the SQL Database;
4. Spoofing of Source Data Store Device (Category: Spoofing), on the UDP data flow between RFID Tag and RFID Reader;
5. Spoofing of Destination Data Store Device (Category: Spoofing), on the UDP data flow between RFID Tag and RFID Reader;
6. Spoofing of Source Data Store GPS receiver (Category: Spoofing), on the UDP data flow between GPS receiver and SQL Database;
7. Spoofing of Destination Data Store SQL Database (Category: Spoofing), appeared three times on the UDP data flow between GPS receiver and SQL Database and on the SQL Database;
8. Spoofing of Source Data Store RFID reader (Category: Spoofing), on the UDP data flow between RFID Reader and SQL Database;
9. Spoofing of Source Data Store SQL Database (Category: Spoofing), on the SQL Database;
10. Web Server Process Memory Tampered (Category: Tampering),on the HTTPS data flow between Web Server and Web Application;

11. Elevation Using Impersonation (Category: Elevation of Privilege), appeared three times on the HTTPS data flow between Web Server and Web Application and on the HTTPS data flow between Human User and Web Application;
12. Spoofing the Human User External Entity (Category: Spoofing), on the HTTPS data flow between Web Application and Human User;
13. Potential SQL Injection Vulnerability for SQL Database (Category: Tampering), on the SQL Database;
14. Potential Excessive Resource Consumption for Web Server or SQL Database (Category: Denial of Service).

Subject 6:

1. Spoofing of Destination Data Store SQL Database (Category: Spoofing), appeared three times on the data flow Authentication and the data flow Data transfer between RFID reader and SQL Database, and on the data flow SQL request between Web Service and SQL Database;
2. Possible SQL Injection Vulnerability for SQL Database (Category: Tampering), appeared three times on the SQL Database;
3. Authenticated Data Flow Compromised (Category: Tampering), appeared one time on the data flow Authentication between RFID reader and SQL Database;
4. Spoofing the Browser External Entity (Category: Spoofing), appeared one time on the data flow HTTPS between Browser and Web Service;
5. Elevation Using Impersonation (Category: Elevation of privilege), appeared one time on the data flow HTTPS between Browser and Web Service;
6. Potential Excessive Resource Consumption for Web Service or SQL Database (Category: Denial of Service), appeared one time on Web Service or SQL Database.

Subject 7:

1. Spoofing of Source Data Store RFID system (Category: Spoofing), appeared one time on the Generic Data Flow between RFID system and SQL Database;
2. Spoofing of Destination Data Store SQL Database (Category: Spoofing), appeared one time on the Generic Data Flow between RFID system and SQL Database;
3. Authenticated Data Flow Compromised (Category: Tampering), appeared one time on the Generic Data Flow between RFID system and SQL Database;

4. Weak Access Control for a Resource (Category: Information Disclosure), appeared one time on the RFID system;
5. Spoofing of Source Data Store RFID system (Category: Spoofing), appeared one time on the Generic Data Flow between RFID system and External Web Service;
6. Spoofing of Source Data Store GPS reveiver (Category: Spoofing), appeared one time on the Generic Data Flow between GPS receiver and GPS satellites;
7. Spoofing of Destination Data Store GPS satellites (Category: Spoofing), appeared one time on the Generic Data Flow between GPS receiver and GPS satellites.

Titre : Securité par la conception : Une approche basée sur les assets pour réduire le fossé entre les architectes et les experts de sécurité.

Mot clés : Assistance à la sécurité, Architecture et Conception, Patrons d'attaques, Sécurité par la conception, Modélisation des menaces (processus), Identification des biens.

Résumé : On observe un nombre croissant d'attaques informatiques causant des dommages aussi bien sur le plan des données que des matériels avec potentiellement à la clé des risques financiers voire humains importants. La sécurité ne peut plus être traitée uniquement comme une exigence dont on ne se soucie qu'une fois un logiciel mis en exploitation dans un contexte technique et organisationnel donné. En effet, un logiciel vulnérable peut être le point d'entrée d'une attaque même dans un réseau sécurisé. Il convient donc de compléter les traditionnelles mesures de protection liées aux infrastructures systèmes et réseaux par un accroissement du niveau de sécurité des applications qu'ils hébergent. L'exigence sécurité doit être prises en compte dans toutes les étapes du développement d'un logiciel. L'une des approches les plus importantes pour ce faire est celle de la modélisation des menaces. Cette approche implique la collaboration lors de séances de brainstorming de plusieurs acteurs et tout particulièrement les architectes logiciels et les experts en sécurité. Car si les architectes connaissent bien les artefacts de conception qu'ils manipulent et les bonnes pratiques d'assemblage, ils ne sont pas toujours bien formés en matière de sécurité. Il n'est malheureusement pas toujours possible d'impliquer à chaque étape d'un développement ces experts en sécurité et cela pour des contraintes de disponibilité, de temps ou de budget. Même quand cette collaboration est possible, en raison du manque de guides, de l'absence d'un processus suffisamment formalisé, ces acteurs éprouvent souvent des difficultés à interagir. Les sessions de brainstorming sont donc souvent menées de manière non optimale et exigent des efforts importants. Le résultat est également fortement dépendant du niveau de sensibilisation et d'expertise en sécurité de tous les acteurs impliqués.

Pour améliorer cette approche et plus généralement la prise en compte au cours de l'étape de conception architecturale de la préoccupation sécurité, nous proposons de définir dans cette thèse une assistance à la sécurité qui s'appuie sur le concept de « biens » (« asset » en anglais). Cette assistance est dédiée aux architectes et les aide à concevoir des systèmes sécurisés même s'ils ne disposent que d'une connaissance limitée en matière de sécurité. Cette assistance les alerte, lors de leur activité de modélisation, des menaces associées au système en cours de conception et peut leur suggérer d'intégrer des mécanismes de contrôle de la sécurité sur les parties vulnérables de ce système dans le modèle d'architecture. L'assistance proposée repose non seulement sur processus clairement formalisé mais également sur une base de connaissance orientée sécurité construite depuis des référentiels internationaux tels que CAPEC et CWE. Le processus permet d'identifier les biens à la fois pertinents et vulnérables, ce qui facilite l'énumération des menaces lors du brainstorming. Nous illustrons ce processus proposé par une étude de cas et montrons l'utilité de ce processus pour faciliter l'énumération des menaces et améliorer les processus de modélisation des menaces existants, tel que SDL de Microsoft. Nous évaluons également notre proposition en menant une expérimentation avec des étudiants qui jouent le rôle d'architectes.

Title: Security by Design: An asset-based approach to bridge the gap between architects and security experts.

Keywords: Security Assistance, Architecture and Design, Attack Pattern, Security-by-Design, Threat modeling (process), Asset-based reference model, Asset identification

Abstract: Nowadays, there is a growing number of security attacks causing damage to both information systems and infrastructures, potentially resulting in significant financial and even human losses. Security can no longer be treated solely as a requirement to be addressed once a software is operational in a given technical and organisational context. Indeed, a vulnerable software can be the entry point of an attack even in a secure network. Traditional protection measures related to system and network infrastructures should therefore be strengthened by increasing the security level of the applications they host. The security requirement must be taken into account at all stages of the software development life cycle. One of the most important approaches for doing this is threat modelling. This approach involves the collaboration of several actors, in particular software architects and security experts, during brainstorming sessions. Actually, even though architects are familiar with the design artifacts they handle, and they have good expertise in assembly practices, they are not always well-trained in security. Unfortunately, it is not always possible to involve these security experts at every stage of the development life cycle due to several constrains such as availability, time or budget limitations. Even when this collaboration is possible, due to the lack of guides and the absence of a sufficiently formalized processes, these actors often have difficulties when interacting. Therefore, brainstorming sessions are often not carried out in an optimal manner and require significant efforts. The outcome is also highly dependent on the level of security awareness and expertise of all the involved actors.

In order to improve threat modeling approaches, and more generally the considerations of security concerns at the architectural design stage, we propose to define in this thesis a security assistance based on the concept of « assets ». This assistance is dedicated to architects and helps them to design secure systems even if they have only limited security knowledge. This assistance alerts them, during their modelling activity, by displaying the possible threats associated to the system being designed, and may suggest a list of possible control mechanisms to mitigate the vulnerable parts of this system at the architectural design stage. The proposed assistance is based not only on a clearly formalized process, but also on a security-oriented knowledge base built upon international references such as CAPEC and CWE. The process enables the identification of both relevant and vulnerable assets, which facilitates threat enumeration during the brainstorming sessions. We illustrate the proposed process with a case study and show its usefulness in facilitating threat enumeration and improving the existing threat modelling processes such as Microsoft's Security Development Lifecycle (SDL) process. Moreover, we evaluate the proposed approach by conducting a crossover experiment with master students in software engineering, playing the role of architects.