# An Asset-Based Assistance for Secure by Design

Nan Messe, Nicolas Belloir, Vanea Chiprianov, Jamal El-Hachem, Régis Fleurquin, Salah Sadou

# An Asset-Based Assistance for Secure by Design

Nan Messe*, Nicolas Belloir*, Vanea Chiprianov*, Jamal El-Hachem*, Régis Fleurquin*, Salah Sadou*
* *Université Bretagne Sud - IRISA, France*
Email: firstname.lastname@irisa.fr

***Abstract*—With the growing numbers of security attacks causing more and more serious damages in software systems, security cannot be added as an afterthought in software development. It has to be built in from the early development phases such as requirement and design. The role responsible for designing a software system is termed an "architect", knowledgeable about the system architecture design, but not always well-trained in security. Moreover, involving other security experts into the system design is not always possible due to time-to-market and budget constraints. To address these challenges, we propose to define an asset-based security assistance in this paper, to help architects design secure systems even if these architects have limited knowledge in security. This assistance helps alert threats, and integrate the security controls over vulnerable parts of system into the architecture model. The central concept enabling this assistance is that of *asset*. We apply our proposal on a telemonitoring case study to show that automating such an assistance is feasible.**

***Index Terms*—Security Assistance, Architecture and Design, Attack Pattern, Secure-by-Design**

## I. Introduction

In the highly interconnected, ubiquitous computing world of today and tomorrow, cyber-security has become a major concern for organisations. To avoid harmful damages with ever escalating attacks, designing security-aware software systems is essential [21]. The role responsible for designing a software system is termed an "architect", knowledgeable about the system architecture design, but not always well-trained in security [8], [17]. Thus they may not adequately deal with security aspects when designing systems. Moreover, it is suggested to bring together in the design team other participants, having different backgrounds and expertise than architects [34], including domain experts and security experts. However, involving security experts into the system design is not always possible due to time-to-market and budget constraints [8], [28].

To allow the modeling of secure systems at early development phases, security modeling languages, such as SecureUML [18] and UMLSec [15], are proposed. In addition, risk assessment methodologies, such as NIST SP 800-30 [3] and OCTAVE [5], also provide the possibility of integrating security aspects into the system development. Nevertheless, these methodologies are time-consuming and require security expertise. Architects without significant security knowledge can not effectively rely on these methodologies to deal with security aspects during the design phase.

In contrary, attackers are creative, actively collaborate and have powerful tools. For example, in 2017, a worldwide cyber-attack 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 [9]. One month later, another ransomware, *Petya*, also used the SMB network spreading techniques to propagate among Windows-based computers even if Microsoft has released patches against *WannaCry* [27]. In 2019, *BlueKeep*, which is a vulnerability in Microsoft Remote Desktop Protocol (RDP) [30], has been estimated to have the same disruptive potential as *EternalBlue*. As we can see, with the need of security grows for applications, services and technologies, the threats also grow as attackers get more organized and sophisticated. As 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 example, ransomwares emerge unceasingly with different attack forms on different specific targets such as SMB and RDP, but the thing in common of these attacks 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, a good general-purpose solution is required on which flexible and adaptable security features can be added. 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 understand the root cause of security breaches hidden behind various attack technique forms, thus helping enhance the security level by design.

To deal with the issues mentioned above, which are i) the knowledge gap between architects and security experts; ii) the high variability of applicable threats over time, depending on the system deployment context and on other aspects, and iii) the lack of a shared terminology for the characterization of cyber security, we propose in this paper a security assistance, which enables architects, who are not always security specialists, to integrate security aspects into the architecture model during the design phase. This security assistance allows to highlight known vulnerabilities contained in the architecture elements and to recommend security countermeasures to architects. The main objective of this paper is to show that it is possible to define and automate such an assistance.

The paper is structured as follows: Section II presents the background and useful knowledge bases required for the security assistance, and a motivating example to illustrate our approach throughout the paper. In Section III, we present

the overview of our approach by introducing an asset-based three-view security assistance framework. In Section IV, we introduce a data model before detailing the assistance process in Section V. Then in Section VI, we verify that this data model and this process provide the possibility of the assistance automation with the motivating example. Related works are discussed in Section VII. Finally, we conclude the paper and discuss future works in Section VIII.

## II. Background and Motivating Example

This section firstly presents the required knowledge background for the security assistance. Then, three well-known public repositories are identified, helping the integration of security knowledge into our assistance using standardized IT elements representation. Thirdly, we introduce a case study which illustrates our approach throughout the paper. Based on this case study, we identify the requirements to which such an assistance should conform.

### A. Security Assistance Knowledge Background

Security aspects include at least the following concepts [29]: i) an *attack* is the realization of a threat that impacts computational resources; ii) a *vulnerability* is a weakness that can be exploited by *attacks* to violate the system security policy; iii) a *control* can be either an alert of vulnerabilities (a residual level of risk to the assets), or a countermeasure (a preventive measure against *attacks*). A security assistance thus requires some forms of knowledge base that encodes at least existing *attacks*, *vulnerabilities* and *controls*. which are fundamental concepts in computer security [29]. The concept of *asset* is also fundamental. An *asset*, as defined by ISO 21827 [1], is "anything that has value to an organisation", such as software, hardware, network, etc. We will later propose to rework on the notion of *asset* to deal with security aspects. Finally, iv) an *IT products naming scheme* is necessary to link, in a standardized and structured way, the assistance with the general IT products that can be involved in different domain architectures.

### B. Useful Knowledge Databases

The *Mitre Corporation*[1] offers security resources and defines a standardized way to refer to IT products. The above information can be retrieved from Mitre knowledge bases.

Firstly, to build secure systems, it is important to think like an attacker and find out vulnerable architecture element. Accordingly, understanding the attacker's viewpoint is critical to build secure systems. Attack pattern [24] is a structured mechanism to capture and communicate this viewpoint including common approaches used by attackers to target weaknesses of software systems. Mitre's Common Attack Pattern Enumeration and Classification (CAPEC) [22] provides a catalogue of common attack patterns that help understanding how attackers exploit weaknesses. It involves a wide range of attack categories, from social engineering, to software and physical attacks. CAPEC helps us retrieve the information about high-level attack goals and attack tactics, which are useful for the security assistance. Secondly, Mitre's Common Weakness Enumeration (CWE) [23] collects common weaknesses that can be exploited by attack patterns in CAPEC. CWE contains the vulnerability information for our assistance. Thirdly, security controls aim at reducing risks caused by vulnerabilities and preventing attacks. Information about security controls can be retrieved from both CAPEC and CWE. An assistance could integrate this security control information to warn the architect about vulnerabilities and to recommend security countermeasures to prevent attacks.

Moreover, the use of CAPEC and CWE uncovers a constraint on the Archirecture Description Language (ADL) and its uage. In particular, to use the assistance in multi-domains, a common and formalized description of IT products is required, to allow automatic machine interpretation and processing. Mitre's Common Platform Enumeration (CPE) [25] provides this possibility by offering a structured naming scheme for IT products. Consequently, it should be feasible to associate each architectural element with a "type" that can be referred to an "element" in the CPE repository. 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 via naming convention rules on architectural elements.

To summarize, a security assistance should integrate information about: i) *attack pattern* (e.g. CAPEC) to represent the attackers' viewpoint, and to retrieve common attack goals and tactics; ii) *vulnerability or weakness* (e.g. CWE) that can be exploited by attackers; iii) *security control* that can be retrieved from mitigation information corresponding to attack pattern and weakness; and iv) *standard naming scheme* (e.g. CPE) to identify general IT elements in a standardized way. Note that CAPEC, CWE and CPE make references one to another, which ease the knowledge integration into the assistance. Another useful knowledge base is the Mitre's Common Vulnerability Enumeration (CVE). We don't need it in this paper because the vulnerabilities in CVE are platform- and technique-specific, which is out of the concern of our proposal.

### C. Motivating Example

To illustrate our approach throughout the paper, we introduce a telemonitoring system for the heart failure management. This system enables communications among the home care point, the hospital and the doctor's remote office in order to reduce the burden of hospitalisations. For our example, we consider that the architect uses an ad-hoc ADL for the architecture modeling. We also assume that the typing of each architectural element is compatible with CPE: either by naming rules or through typing mechanisms expressible in the ADL, which are not in the scope of this paper.

As shown in Figure 1, the telemonitoring system is mainly composed of three parts: i) a home care point with a subset of components such as a pacemaker, wearable technologies and a fix phone; ii) a healthcare center with components such as

[1]The Mitre Corporation : https://www.mitre.org/

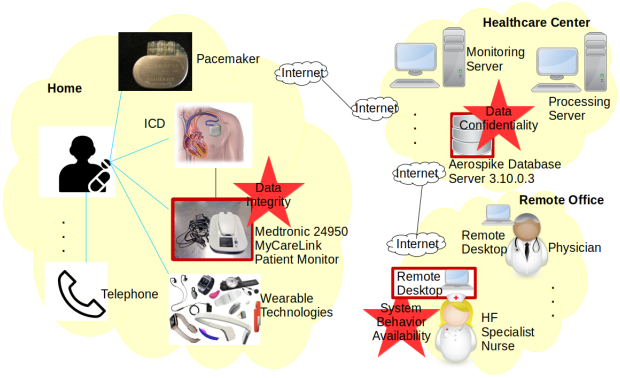


Figure 1: Heart Failure Patient Health Telemonitoring System

monitoring and processing servers and iii) a doctor's remote office with remote desktops.

We focus on the heart rhythm monitoring scenario to illustrate how the assistance helps integrating security aspects 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 and 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.

This 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) 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 of their desktop is not available.

### D. Security Assistance Requirements

To prevent such incidents, a security assistance could be helpful during the architecture modeling phase, even in the presence of abstract architectural elements. Indeed, the architecture design can be performed in several stages and therefore it involves elements of different levels of abstraction. It is possible to mix abstract elements (whose implementation details are not specific yet) with concrete ones (which are precise). The architecture is thus refined gradually to become finally a concrete architecture model. During this activity, the architect should be able to request the launch of the security assistance at any time. 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 bases.

The trigger behind this assistance is that architects have limited security skills in general [8], [17]. However, to allow our assistance, architects have to at least be able to: i) distinguish the elements they want to protect (assets) in the architecture model and ii) 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 and general security knowledge bases, our security assistance returns alerts about the architectural elements' vulnerabilities and recommends countermeasures to the prevent attacks.

In Figure 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" 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 the security properties that need to be assured for each chosen 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 1). To help ensure these properties, the assistance could interrogate a 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 design to change the default credentials in order to prevent the corresponding attacks.

## III. The Foundations of the Security Assistance

In this section, we present the basis of our security assistance approach. We refine the concept of *asset* into *domain, vulnerable and vulnerable domain asset* to allow the definition of the security assistance. Then we elaborate an asset-based three-view framework and we introduce "Attack Pivot Tree"

(APT), which is designed based on "attack tree" and focuses on asset-based security analysis, to link up these three assets.

### A. Rework on the Refinement of Asset

The traditional process of risk assessment begins by 1) listing valuable assets of an organization, 2) identifying threats to these assets and vulnerabilities exploitable by these threats, 3) estimating risks to these assets and 4) proposing countermeasures to protect them. Therefore, *asset* is a central and pivotal concept. Asset analysis is a critical step for risk assessment, involving the viewpoints of several actors, such as the architect, the attacker (tester) and the security expert (defender). In traditional approaches, these actors work together to identify assets which are useful for further risk assessment steps. In our opinion, *asset* means differently for these three actors. It thus encompasses three viewpoints: i) **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; ii) **the attacker's viewpoint**, illustrates mainly a vulnerable architecture element as the target of an attack with the following information: attacker's tactic, exploitable vulnerabilities, security breaches and their impact; iii) **the security defense expert's viewpoint**, who understands how the attacks are performed and proposes security control solutions such as vulnerability notifications and countermeasures to prevent attacks.

A same asset may mean differently under different viewpoints. For example, "Medtronic monitor" 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 *asset* according to these three views to facilitate the fulfillment of the security assistance:

**Domain Asset (DA):** Anything that has value to the architect, towards the fulfilment of the function and goal of the system. It thus represents assets that are domain specific. For example, the "database server" is a valuable domain asset to keep 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 asset. In contrast to domain assets, which are anchored to architecture elements, vulnerable assets enable raising the security-based abstraction level. This allows tapping into reusable attack knowledge patterns, which do not depend on the specific architecture model under design. As such, domain assets comprises 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 weaknesses. By distinguishing these two assets,

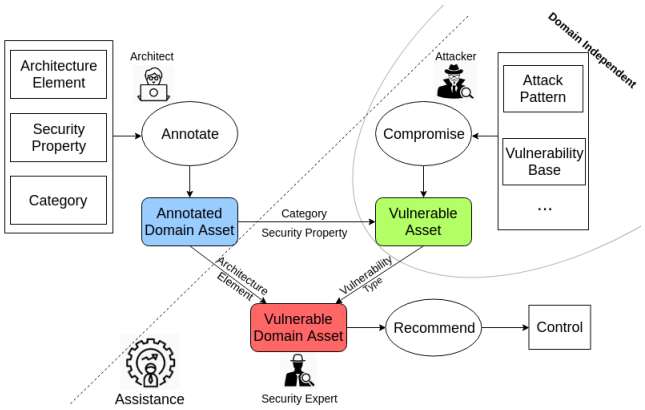


Figure 2: Asset-Based 3-View Security Assistance Framework

the 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).

**Vulnerable Domain Asset (VDA):** Anything that has value to the security defense expert. As the vulnerable asset operates at a high abstraction (pattern) level, we introduce the VDA as its projection (instance) on the architecture model. The security expert is aware of the value of the VDA for the architect and of its vulnerabilities exploitable by potential attacks and thus proposes countermeasures.

Therefore, we consider *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 3 different viewpoints from the architect, the attacker and the security defense expert, which helps us refine the notion of *asset* in order to provide a security assistance to the architect.

### B. Asset-Based Three-View Security Assistance Framework

In the architect's viewpoint as presented in Figure 2, a DA 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 1), with a security property to be preserved (e.g., *confidentiality, integrity* and *availability*) and with a DA category (*data* and *system behavior*).

In the attackers' viewpoint, the goal is to compromise a VA using their knowledge of vulnerability types (e.g., contained in databases such as CWE) and of attack patterns (e.g., captured in databases like CAPEC). In contrast to DA, which concern with architecture elements, VA enables raising the security-based abstraction level. They represent general direct attack targets, whose compromise can indirectly harm related DAs. In our approach, VAs are retrieved from common security knowledge base (e.g. CAPEC). These VAs are characterized by vulnerabilities, which we retrieve from the database CWE.

To define the mapping relation between DA and VA, we use 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 another hand, the security property of a category can be impacted by the compromise of VAs.

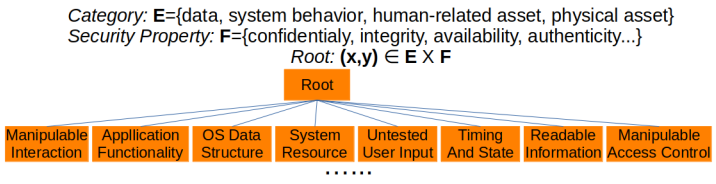


Figure 3: The Attack Pivot Tree (APT)

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 knowledge base to find the compromises of which VAs have negative impacts on the *Data Integrity*. In our example, the assistance finds for example the VA "readable credentials". If "readable credentials" are stolen 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.

The defender's viewpoint takes into account the vulnerabilities of the assets that can be exploited by attacks. To protect the assets from being compromised, the defender recommends security countermeasures. In order that the security assistance takes the role of the security defense expert, it is necessary to relate the concept of VDA with those of DA and VA.

If for a DA, the assistance process 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 readable credential" is a VDA.

VA and VDA (together with other security knowledge such as vulnerabilities and controls) depict *assets* from the attacker and defender viewpoints. This information is usually not easily-understandable to the architect. Therefore, our assistance encodes the knowledge of these two viewpoints, taking their roles and helping the architect with recommendations. The assistance achieves this by bridging the three viewpoints through the refinement of the concept of *asset*.

### C. Relation Between DA and VA : The Attack Pivot Tree

As 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 VDA. Attack Pivot Tree (APT) is used to allow this transition.

APT is designed based on the *Attack Tree* [16]. An attack tree 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 actions that allow an attacker reaching the root are iteratively represented as branches and intermediate level nodes. Each node defines an action-based subgoal, and each subgoal may have its own set of further subgoals. The bottom nodes on the paths, i.e., the leaf nodes, represent different actions to initiate an attack. Each node other than a leaf is either an AND-node or an OR-node. To achieve the goal represented by an AND-node, the subgoals represented by all of that node's subnodes must be achieved; and for an OR-node, at least one of the subgoals must be achieved. Our motivation of the use of attack trees is to effectively exploit the information available on attack patterns in CAPEC.

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. 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 the architect's viewpoint. 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 CAPEC (*Category* and *Meta Attack Pattern*), 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.

In APT, to enable the link between DA and VA, the root of the APT is a 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 3 presents 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 databases CWE) and to 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 controls. The APT thus enables finding for an architecture element possible vulnerabilities and controls to assist the architect.

For example, in Figure 3, based on our security knowledge, we manually identify and extract the 8 children (VAs) of the root node (summarized in Table I), from the most abstract level of the attack mechanisms in CAPEC.

According to the above philosophy, we propose an approach which integrates and structures the necessary knowledge into a data model, presented in the next section. We propose 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 I: VAs from Figure 3 extracted from CAPEC

sequence of tasks organised in a systematic process based on this data model to enact the assistance (Section V).

## IV. Data Model

Based on the asset-based three-view security assistance framework, we introduce the assistance data model capturing important concepts and their relations in this section. For the simpler readability, we split it into four parts, including the three viewpoints, represented with different colors: the domain architecture (blue), the attack (green) and the defense (red). The fourth transverse part describes the structure and refinement mechanisms of concepts presented in the three viewpoints. To ensure that the relations between these four parts are preserved, some concepts are reused in several parts.

### A. Domain Architecture Specific Aspects

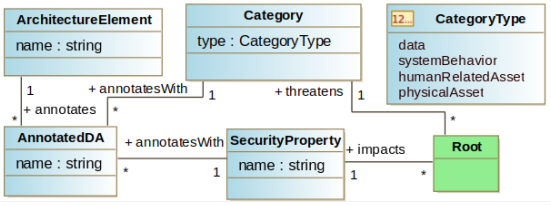


Figure 4: Domain Specific Architecture

The concepts described in Figure 4 capture the specificities of the domain architecture under study (e.g., healthcare), which are required to launch the security assistance.

An *AnnotatedDA* annotates a domain asset, which is an *ArchitectureElement*, with a *SecurityProperty* and a *Category*. As presented previously, the *SecurityProperty* and *Category* are used to relate the *AnnotatedDA* to the *VA*, through the concept of *Root*. The compromise of the *Root*, in its characteristic as the final goal of the APT (cf. Sec. III-C), *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 [2]. The architect is supposed to understand these properties and annotates them on the *ArchitectureElement* to require the preservation of these properties. A *Category* represents the category of the *AnnotatedDA* to be protected. Based on assets that are commonly impacted by security attack consequences, we identified 4 types of categories. Indeed, most of threats in [29] 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. identity. Finally, the destruction of physical devices is also a possible consequence. Hence *CategoryType* contains: *data*, *systemBehavior*, *humanRelated* and *physicalAsset*.

### B. Attack Specific Aspects

This subsection captures the attacker's viewpoint as shown in Figure 5. 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 *VA* and *Root*. As APT is centered on *asset*, which is pivotal to our approach, we name the node of the APT: *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*. It consists of exploits performed by an attacker, to take advantage of *VulnerabilityTypes* to obtain negative impacts. A *VulnerabilityType* models a vulnerability, a weakness or a design error that may result in an undesirable event.

### C. Defense Specific Aspects

The concepts discussed in this subsection capture the viewpoint of the security defense expert as shown in Figure 6. An *ArchitectureElement* can be matched to 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*. 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 security defense expert's viewpoint. The concept of *ElementType* encodes information about possible elements of an architecture. Our knowledge base contains a list of such *ElementTypes* and their relations. This allows the

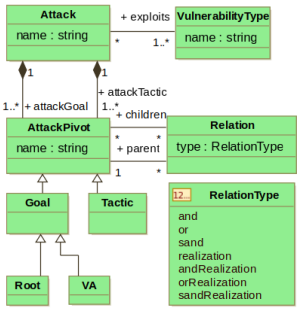


Figure 5: Attack Specific Aspects

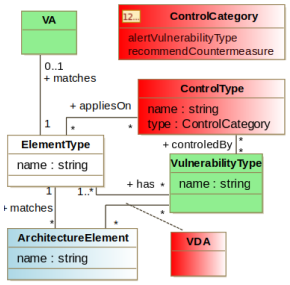


Figure 6: Defense Specific Aspects

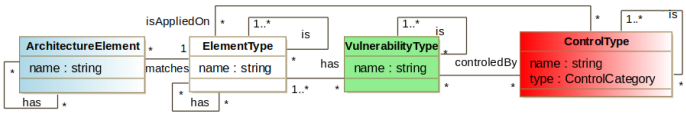


Figure 7: Refinement and Structural Mechanisms

automatic assistance process 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 V, the matching algorithm uses the naming information. Therefore, a common naming scheme is essential. This imposes 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. 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.

### D. 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 7. 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 that not all vulnerabilities of specific product are revealed in the security breach databases (not exploited by any attack yet), but it doesn't mean 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 database CVE. However, there are more abstract vulnerability types, such as those proposed by databases 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 the 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 the component.

## V. Security Assistance Process

The data model unifying concepts from the 3 viewpoints allows us to integrate (semi-)automatically the knowledge specific to attacker and defender into our assistance. In this section we present in detail how the security assistance process assist the architect based on this data model.

The security assistance process is presented using an enhanced BPMN diagram as shown in Figure 8. It consists of two major phases: one performed by the architect, and another performed by the automatic assistance process. 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 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. 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 uses the APT to *develop (refine and/or decompose) tactic* (9). The root of the APT is the *SecurityProperty* and the *Category* selected by the architect. These two tasks result in two lists of *Selected VA* and respectively of *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 matching with an item in the *Selected ElementType* list, then the assistance *highlights* the corresponding *ArchitectureElement* as a *VDA* (12). The *ElementType* that

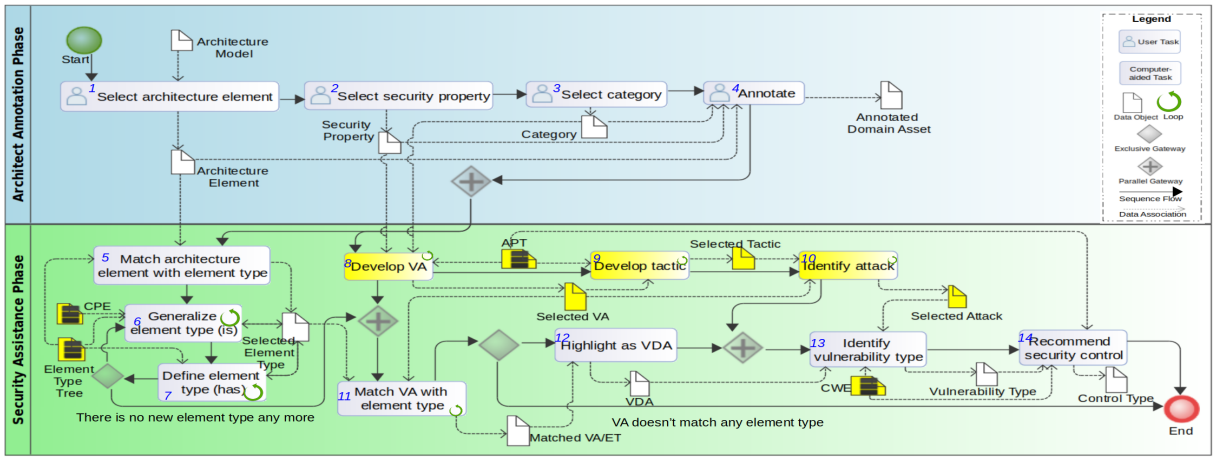


Figure 8: Assistance Process. Each task is uniquely identified with a number placed before its description.

is related with the *ArchitectureElement* from the architecture model is as valuable as the *Annotated DA* for the architect. 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 security defense expert's viewpoints.

Using the identified *VDA* and a vulnerability database (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).

## VI. Application of the Security Assistance on the Motivating Example: The Results

To show the automation and usefulness of the security assistance, we apply it on three annotated DA in the motivating example to illustrate concrete and abstract aspects. We first show how we 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.

### A. Assistance Enactment

As the data model 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 are implemented as "SQL select statements". Our database is filled with information extracted from several existing, widely-used databases. As such, the *VAs*, *Tactics* and *Relations* of the APT, are extracted from CAPEC. 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, manual, and based on security experts' knowledge. In the current state, our database contains a part of the knowledge of the existing databases. For example, our database contains the knowledge associated with 50 attack patterns from the 575 that CAPEC contains, and with 62 vulnerabilities out of the 1141 that CWE contains.

### B. Assistance on DA1: Concrete Security Control Recommendation on a Concrete Architecture Element

The results of the application of the assistance process on these three *Annotated DAs* are presented in Table II. Each row corresponds to each *Annotated DA*. Each column corresponds to data objects obtained from the 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 that the architect *selects* the *ArchitectureElement* "Medtronic monitor" (the task *1 Select architecture element* in Figure 8). The architect also *selects* (2) the *SecurityProperty* "Integrity" and *selects* (3) the *Category* "Data", with which he/she *annotates* 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".

In parallel, starting from the *SecurityProperty* "Integrity" and the *Category* "Data", considered as the *Root* of the APT, 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 (9,8), the assistance *identifies attacks* (10), such as "Hard-coded password realized by reading sensitive strings

| (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 II: Assistance results for the three cases of the motivating example.

within an executable".

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 assistance highlights at least the "Medtronic Monitor Hard-coded Password" as a VDA, a concrete constituent of the concrete architecture element "Medtronic monitor"; alerts the architect to at least the *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".

### C. Assistance on DA2 and DA3: Abstract Security Controls on a concrete and respectively Abstract Architecture Element

The application of the 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 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 automatize the assistance process on a database that we have defined. A prototype web application is implemented to show the feasibility[2] of this assistance.

[2]http://share-irisa.univ-ubs.fr/abs4sos/index.php

## VII. Related Work

A number of concepts of our data model are inspired from risk assessment/management approaches: the initial definition of *Asset* (central concept to risk assessment, which we refined into the *DA*, *VA* and *VDA*), *VulnerabilityType* and *Control*. Some of these concepts can be found in other model-based security approaches, such as attack trees or modeling languages for describing security defensive architectures. We discuss the relation of our work with these existing approaches.

**Asset-based risk assessment:** There are numerous risk assessment methods, reviewed and compared for example by [13]: 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 into primary and supporting, or ISSRM into intangible business and tangible information system. [32] proposes 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, which we do with the concept of *VA*. Moreover, these risk assessment methods require long application times and the intervention of security experts. Our approach shows the feasibility of integrating automatically security knowledge, thus enabling risk assessment more easily.

**Model-based security engineering:** Model-based approaches cover risk assessment, attacker modeling and defensive architectures [6]. For example, Coras, Magerit and Mehari are model-based risk assessment methods. Formal models such as attack trees [19] and Petri nets [26] model concrete attack paths on concrete assets, and they need the intervention of security expert. Compared to an attack tree, our attack pivot tree is more generic, asset-based, includes common attack goals and tactics and easily relates to architectural elements.

Security architectures are usually described and analysed using modeling languages, such as SecureUML [18] and UMLSec. Twenty-eight of these languages are reviewed by [33]. This systematic review identifies, among other limita-

tions, that only few (6 out of the 28 surveyed languages) propose automatic analysis mechanisms. Four case studies are discussed in [10] using a strategic, system-wide architectural approach, implemented as a security framework. Guidelines are proposed to detect security design flaws in [31]. Both of them lack an automatic process as well. Our approach addresses this limitation by offering the possibility of defining an automatic security assistance for architects.

Our approach thus covers several limitations of the existing state of the art. It bridges system domain specific architecture design with security attack and defense engineering (as suggested for example by [14] and [7]).

## VIII. Conclusion and Future works

In this paper, we advocate the definition of a security assistance for the architect when designing software systems. We propose an asset-based three-view security assistance framework, which includes: i) a data model and ii) a systematic process. We rely on sound existing security knowledge bases, such as CAPEC and CWE, to build the assistance database implementing the data model. We show that we can automatize this assistance, by applying it on a telemonitoring case study. This application also highlights how the proposed assistance can help the architect when integrating security aspects into the early phases of software development life cycle.

In the future, we plan to: 1) Add other concepts and related mechanisms such as *Risk* and its computation, for which at least partial information could be extracted from existing bases, such as attack impact from CVSS [20]. 2) Automatize the security knowledge extraction from existing security bases and its insertion into the assistance database. Possible directions include machine learning approaches such as topic modeling [4]. 3) 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 [12]. 4) Develop a tool suite which enacts the assistance during the architecture modeling phase, relying on SysML for example. 5) Extend concepts such as *SecurityProperty* (e.g., to take into account authenticity or traceability), *RelationType* (e.g., with XOR or NOT). This requires taking into account future development of existing databases (e.g., CAPEC may consider other security properties). 6) Consider the inter-dependencies among domain assets and analyze the consistency of the architect's annotations, using for example colored graphs [11] to identify contradictions.

## References


[1] ISO/IEC 21827:2008. Information technology – security techniques – systems security engineering – capability maturity model, 2008.

[2] ISO/IEC 27000:2018. Information technology - security techniques - information security management systems - overview and vocabulary.

[3] NIST SP 800-30. Guide for conducting risk assessments.

[4] S. Adams, B. Carter, C. Fleming, and P. A. Beling. Selecting system specific cybersecurity attack patterns using topic modeling. In *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications*, pages 490–497, Aug 2018.

[5] Ch. J. Alberts and A. Dorofee. *Managing Information Security Risks: The Octave Approach*. Addison-Wesley, USA, 2002.

[6] S. Alpers, R. Pilipchuk, A. Oberweis, and R. 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*, pages 109–124. Springer, 2019.

[7] S. Bode, A. Fischer, W. Kühnhauser, and M. Riebisch. Software architectural design meets security engineering. In *16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 109–118, April 2009.

[8] V. Casola, A. De Benedictis, M. Rak, and U. Villano. A novel security-by-design methodology: modeling and assessing security with a quantitative approach. *Journal of Systems and Software*, 2020.

[9] Microsoft Security Response Center. Guidance for wannacrypt attacks, 2017.

[10] H. Cervantes, R. Kazman, J. Ryoo, D. Choi, and D. Jang. Architectural approaches to security: 4 case studies. *Computer*, 49(11):60–67, 2016.

[11] E. Coatanéa and R. Roca. Dimensional analysis conceptual modeling supporting adaptable reasoning in simulation-based training. In *13th Conference on System of Systems Engineering*, pages 245–252, 2018.

[12] Hamish Cunningham. Gate, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002.

[13] D. Gritzalis, G. Iseppi, A. Mylonas, and V. Stavrou. Exiting the risk assessment maze: A meta-survey. *ACM Comput. Surv.*, 51(1):11:1–11:30, January 2018.

[14] R. A. Jones and B. Horowitz. A system-aware cyber security architecture. *Syst. Eng.*, 15(2):225–240, June 2012.

[15] J. Jürjens. Umlsec: Extending uml for secure systems development. In *UML 2002 — The Unified Modeling Language*, 2004.

[16] R. Kumar, S. Schivo, E. Ruijters, B. Yildiz, D. Huistra, J. Brandt, A. Rensink, and M. Stoelinga. Effective analysis of attack trees: A model-driven approach. In *Fundamental Approaches to Soft. Engineering*, pages 56–73. Springer, 2018.

[17] Ruby B Lee. Security basics for computer architects. *Synthesis Lectures on Computer Architecture*, 8(4):1–111, 2013.

[18] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *UML — The Unified Modeling Language*, pages 426–441, 2002.

[19] S. Mauw and M. Oostdijk. Foundations of attack trees. In *Proc. of the 8th Int. Conf. on Inf. Secu. and Cryp.*, ICISC'05, 2006.

[20] P. Mell, K. Scarfone, and S. Romanosky. *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. NIST and Carnegie Mellon University, 1 edition, June 2007.

[21] D. Mellado, E. Fernández-Medina, and M. Piattini. A common criteria based security requirements engineering process for the development of secure information systems. *Computer standards & interfaces*, 29(2):244–253, 2007.

[22] MITRE. Common attack pattern enumeration and classification, https://capec.mitre.org/.

[23] MITRE. Common weakness enumeration. https://cwe.mitre.org.

[24] A. Moore, R. Ellison, and R. Linger. Attack modeling for information security and survivability. 2001.

[25] NIST. Common platform enumeration. https://cpe.mitre.org/.

[26] J. L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3):223–252, 1977.

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

[28] Adam Shostack. *Threat Modeling: Designing for Security*. 2014.

[29] W. Stallings and L. Brown. *Computer Security: Principles and Practice*. Prentice Hall Press, 3rd edition, 2014.

[30] TechTarget. Bluekeep (cve-2019-0708), 2019.

[31] K. Tuma, D. Hosseini, K. Malamas, and R. Scandariato. Inspection guidelines to identify security design flaws. In *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, ECSA '19.

[32] G. Tziakouris, M. Zinonos, T. Chothia, and R. Bahsoon. Asset-centric security-aware service selection. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 327–332, June 2016.

[33] A. Van Den Berghe, R. Scandariato, K. Yskout, and W. Joosen. Design notations for secure software: A systematic literature review. *Softw. Syst. Model.*, 16(3):809–831, 2017.

[34] Hans van Vliet. Software engineering - principles and practice. 2007.