Välja *et al. Cybersecurity* (2020) 3:19
https://doi.org/10.1186/s42400-020-00060-8

Cybersecurity

RESEARCH Open Access

Automating threat modeling using an ontology framework

Validated with data from critical infrastructures

Margus Välja[1], Fredrik Heiding[1*], Ulrik Franke[2] and Robert Lagerström[1]

Abstract

Threat modeling is of increasing importance to IT security, and it is a complex and resource demanding task. The aim of automating threat modeling is to simplify model creation by using data that are already available. However, the collected data often lack context; this can make the automated models less precise in terms of domain knowledge than those created by an expert human modeler. The lack of domain knowledge in modeling automation can be addressed with ontologies. In this paper, we introduce an ontology framework to improve automatic threat modeling. The framework is developed with conceptual modeling and validated using three different datasets: a small scale utility lab, water utility control network, and university IT environment. The framework produced successful results such as standardizing input sources, removing duplicate name entries, and grouping application software more logically.

Keywords: Threat modeling, Ontologies, Automated modeling, Conceptual models, Ontology framework

Introduction

Threat modeling is a growing trend in cyber security and vulnerability assessments. Creation of holistic models of the weaknesses and vulnerabilities of an organization can provide effective methods of making secure systems (Torr 2005). However, it is not necessarily an easy or straightforward task. Owing to the increasing complexity in information technology (IT) architectures and the rapid increase of digital threats, it is difficult to maintain an up-to-date and comprehensive threat model of a given system (Berger et al. 2013; Moral-Garca et al. 2014). In addition to this, threat modeling is still predominantly a manual task and thus both time consuming and error prone (Ekelhart et al. 2006), particularly because models quickly become outdated (Aier et al. 2009; Aier et al. 2009). Because of this, automation of threat modeling is preferable and we focus on creating automated threat models by using ontologies and conceptual modeling of IT systems.

Furthermore, the cyber security of critical infrastructure is receiving increasing attention every year. As the digitization of industrial complexes is increasing, it is crucial to ensure that the security of these systems is maintained (Krumay et al. 2018). Several studies have been conducted to create new tools to enhance the security of industrial machines, such as new intrusion detection systems specifically tailored to monitor power systems (Pan et al. 2015) or deep learning-based tools for detecting incoming threats (Catak et al. 2019).

To contribute to the security of critical infrastructure, our threat modeling ontology framework was validated against three datasets, each of which is connected to some part of critical industrial systems, as further explained in "Datasets" section.

There have been attempts to automate threat modeling (Barankova et al. 2020; Xu et al. 2012). However, these methods are less precise than they would have been if domain knowledge had been included in the modeling process. For example, if information is obtained from several data sources (e.g., an active directory and a vulnerability scanner), small differences in the representation of

*Correspondence: fheiding@kth.se
[1]KTH Royal Institute of Technology, 100 44 Stockholm, Sweden
Full list of author information is available at the end of the article

software names can result in duplicated information in the model. Another problem for modeling automation is mismatched data granularity, which occurs when the level of data abstraction of a data source is different from that of the model-to-be-created (Farwick et al. 2013).

Ontologies offer a promising method to solve these problems. In general, ontologies are designed to solve problems or answer domain-specific questions (Gruber 1995). This is achieved by supporting computations with structured data to provide consistency and entailment. Ontologies can be used to address various data-quality problems at run time by having controlled vocabulary concepts and machine-processable semantics (Maedche and Staab 2001a).

To improve the quality of ontologies, it has been proposed that their design should be based on conceptual patterns (Gangemi and Presutti 2009; Falbo et al. 2013). In Gangemi and Presutti (2009), it was reported that small ontologies with explicit motivations to connect use cases with designs can be used as building blocks for automatic modeling. Moreover, such an approach facilitates the alignment, merging, and reusing of ontologies.

To improve automated threat modeling, we propose an ontology framework for threat modeling. The purpose of this framework is to improve automated threat modeling by addressing the two problems mentioned previously: lack of domain knowledge and mismatched granularity. The ontology based framework is created with conceptual modeling built on general knowledge. The usefulness of the created models is evaluated through the implementation and use of the modeled framework. The proposed framework fits into an automated model-creation process (Fig. 1) that has been implemented separately from this work (Vlja et al. 2019), and the proposed framework is presented using ontology patterns defined in Falbo et al. (2013); Gangemi and Presutti (2009).

The remainder of this paper is structured as follows. "Related work" introduces related works. The ontology framework is explained using ontology patterns in "Threat modeling ontology framework". The same section also details the implementation of the ontology and the knowledge acquisition. The application of the ontology framework is explained in "Case study". The paper is concluded with a discussion in "Discussion and conclusion" sections.

Related work

Here, we list the relevant related literature. This section is presented in four parts: threat modeling, automation of threat modeling, conceptual modeling, and ontologies.

Threat modeling

Threat modeling is a growing trend in cyber-security domain as it can assist in several aspects of making a system more secure, such as clarifying a vulnerability analysis, facilitating decision support, and improving documentation (Torr 2005).

Most studies regarding threat modeling can be categorized into one of three clusters (Xiong and Lagerstrm 2019): focus on the application of the models (Cardenas et al. 2009; Pei et al. 2004), discussion on threat modeling methods (Satnam Singh et al. 2004), or analysis of the threat modeling process (Dhillon 2011; Steven 2010). Most of these studies have been conducted to address manual threat modeling. Automation of the models has been focused in very few studies (Xiong and Lagerstrm 2019). In these few studies, ontologies are not a vital part.

Threat modeling is growing in both industry and academia (Akhawe et al. 2010). However, organizations sometimes avoid implementing threat modeling because it is a resource-intensive and time-consuming task. Because of this, there is a need for more modeling tools and frameworks to make modeling easier and more widely accessible to security employees (Steven 2010).

Threat models can be implemented and tailored for different domains and industries such as smart grids and the energy sector (Jiang et al. 2014), where they can enhance the security of Advanced Metering Infrastructure to protect against energy theft.

Mathematical modeling enables threat modeling to map security threats, categorize threat levels, and clarify intended functions, as well as provide ways to mitigate threats and improve the general security architecture (Xu and Nygard 2006).

Domain-specific attack languages can be used to make the creation of new attack graphs and threat models cheap and efficient (Johnson et al. 2018). Such languages can clarify the reasoning of a generic attack in a specific domain and facilitate the modeling or instantiating of a specific system in that domain. Domain-specific languages can be created using the Meta Attack Language, which provides a way to formalize the structure and input of specific languages (Johnson et al. 2018).

Automated threat modeling

Formalization of input data is a natural part of model automatization and facilitates the automatic creation of threat models. The need for manual editing of input data when creating threat models is reduced by formalizing current threats and vulnerabilities, as well as adapting them to the FSTEC threat data bank structure (Barankova et al. 2020).

Tiramisu is a software that can be used to automate threat modeling via attack path analysis. Security threats are quantified by weighing the severity of potential attack paths, and the severity of vulnerabilities are calculated based on the ranking from famous severity databases and manual evaluation of potentially lost business value (Chen et al. 2007).

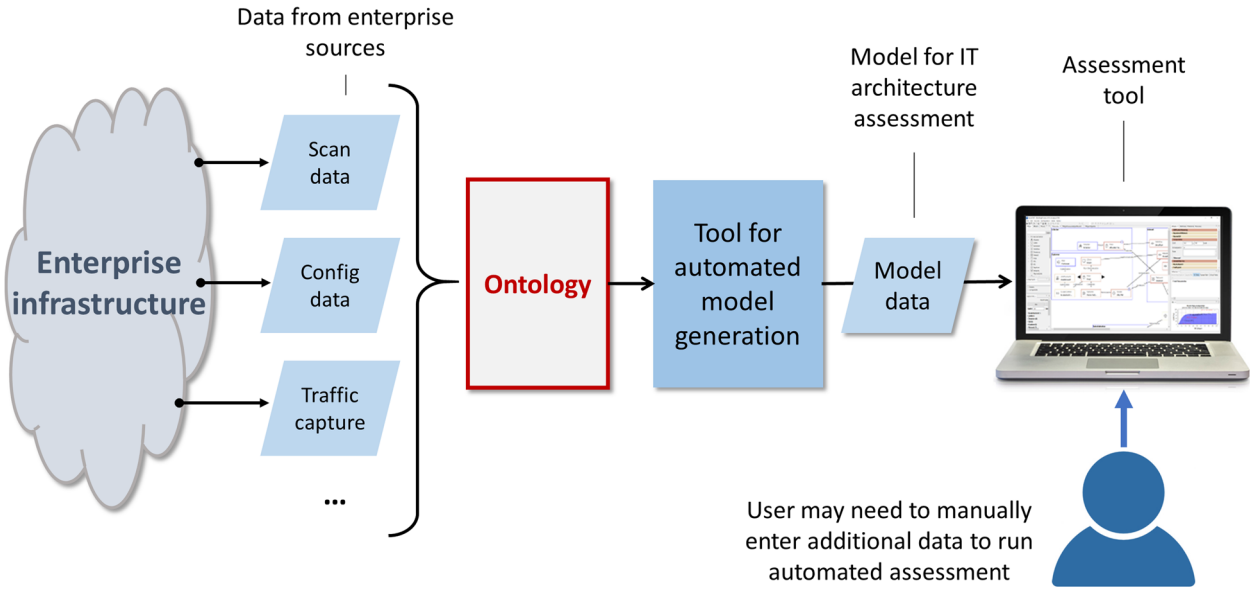

Fig. 1 Data is collected from an enterprise environment, standardized, merged, and finally transformed into a threat model

A number of studies have been conducted to automate security tests using threat models, but the focus has mainly been on using the threat models to achieve automation of the test, rather than automating the creation of the models. In Xu et al. (2012), a method was proposed to represent threat models with Petri nets and use these to derive attack paths that can be converted into executable test code. This enabled partial automation of the security testing but the creation of threat models remained manual.

Microsoft's SDL Threat Modeling Tool offers a way to automatically create data flow diagrams for security analysis of a system; however, the ways to automatically populate the data for the diagrams and formalize input requirements are limited (Kornecki and Janusz 2015).

Another way to automate security testing is to generate automatic attack steps depending on the systems architecture, vulnerabilities, and relevant threats. However, similar to the previous case, automation is only applied to the security testing and not to the model creation (Marback et al. 2013). In Marksteiner et al. (2019), a method to use threat models was demonstrated to automate security tests towards Industrial Internet of Things devices. In Patil and Pawar (2012), web applications were targeted, future possibilities were discussed, and several problems were identified that must be solved to achieve automated model creation for fully automated testing. The requirements are predominantly aligned with what we propose to solve with ontologies, such as the need for a formalized template for the node structure in the model, as well as a standardized and formalized way to populate the values of the model using the input data.

Conceptual modeling

The goal of conceptual modeling is to improve our understanding of a given problem and design better systems. These systems can be particular software systems or larger systems-of-systems representing architectures (Axelsson 2015).

Modeling is generally at the heart of enterprise disciplines such as systems architecture (Johnson et al. 2007; Lagerström et al. 2010) and enterprise architecture (Johnson et al. 2014; Lankhorst et al. 2004).

In the enterprise context, models are needed to handle the increasing complexity of IT-landscapes. Automation of modeling is desired but is hindered by numerous problems. In Rahm and Do (2000), heterogeneous data sources were integrated. In Florez et al. (2014), imperfections in enterprise models were studied, and low quality of information was reported as one problem. In a study regarding current and future enterprise documentation, mismatching granularity between the collected datasets and the enterprise model was identified as another problem (Farwick et al. 2013) and (Roth et al. 2013).

Conceptual models represent a domain without implementation details. In a study in which the effect of applying ontology-based modeling rules on modeling decisions was examined, conceptual modeling was explained to occur during the first stages of system development (Soffer and Hadar 2007). According to the authors, conceptual models are qualitative in nature, and thus it is common that different people can create correct but different conceptual models of the same domain. Ontology can be defined as a specification of a conceptualization. Several authors found that ontologies are useful during

the run-time of systems (Guizzardi et al. 2003; Maedche and Staab 2001a), and (Soffer and Hadar 2007). In Vasilecas et al. (2006), it was reported that ontologies tend to be richer and can provide a controlled vocabulary of concepts that are explicitly defined with machine processable semantics. According to the authors, who studied the automatic transformation of ontology into conceptual models, a single ontology could give rise to multiple conceptual models. In Guizzardi et al. (2003), the use of ontologies was proposed to evaluate the ontological correctness of conceptual models. In Guizzardi et al. (2004), a Unified Modeling Language (UML) profile was proposed for representing ontologies and for conceptual modeling. The authors also studied the use of design patterns to solve recurrent problems in conceptual modeling.

Ontologies

Several studies have been conducted regarding the use of ontologies to improve model and data quality. In Jarrar et al. (2003), an ontology engineering framework was presented that can represent ontologies with conceptual modeling approaches. The authors developed a conceptual markup language that can express conceptual diagrams and supports run-time processing. In Antunes et al. (2014), constructs based on description logics were proposed for the integration and analysis of enterprise architecture models. The authors surveyed possible analysis approaches and matched reasoning features of description logics to different types of enterprise architecture analysis.

Data integration is a central part of ontologies, and by integrating large scale datasets into a single coherent system, process data and static data can be combined to a single model while maintaining the identity of domain objects over time. In Cesare et al. (2016), this was achieved with a semantic data integration framework for integrating data in large scale enterprise systems.

Ontologies can also be used to integrate various domain description languages, maintain coherence, consistency, and traceability between the representations of domain languages, and automate the analysis of models. Model analysis techniques are applied to enforce meta-model coherence, conformance, and ontological integration of stakeholder viewpoints (Antunes et al. 2014). Furthermore, in Caldarola et al. (2015), an approach was proposed to integrate ontologies for ontology reuse to achieve information standardization.

Semantic techniques serve as a tool for analyzing enterprise architecture models, where ontologies represent the conceptual models and derive logical conclusions about the models. In Antunes et al. (2016), such an analysis was performed using both SPARQL and computational inference, and the approach was claimed to have facilitated analysis using syntactic and semantic information from the models. The semantics and structure of data can cause organization-wide heterogeneity problems, as described in Song et al. (2013). The authors of that study proposed an ontology driven framework to solve these problems, and according to them, these problems arise when heterogeneous systems are evaluated and redesigned.

Ontologies are also used in more organizational settings such as aligning IT and business processes. In Hinkelmann et al. (2016), a continuous adaption approach was proposed for such an alignment to combine machine interpretable ontologies and enterprise modeling. The ontologies were used to identify adaption needs, and graphical models supported analysts in decision making. Similar to our approach, automated decision support was addressed in that study; however it was only for the continuous alignment of business and IT. In Pittl et al. (2017), the method to use enterprise models was investigated to build an ontology for representing risks and mitigation measures for a system. This knowledge was used to capture logic, recognize structure in the data, and organize information. In Maedche and Staab (2004), an ontology engineering framework was presented for ontology acquisition. The proposed framework supports ontology import, extraction, pruning, refinement, and evaluation.

Some studies have been conducted regarding ontologies for threat modeling. In Ekelhart et al. (2006), threat modeling of corporate assets was focused upon, and in Gong and Tian (2020), the focus was on an ontology to enhance threat models for a cyber range. In Chhaya et al. (2019), the aim was to create a web ontology language and use ontologies to create better threat models for protecting against drones and low, slow, and small unmanned aerial vehicles. However, automation has not been the focus. In Luh et al. (2016), TAON, a so-called APT ontology made to mitigate digital risk by creating threat models, was introduced. However, automation was not focused upon, and it was mainly mentioned as a direction for future research.

We conclude that many applications of ontologies support automated reasoning and ontologies seem to be a suitable way of storing knowledge for threat modeling and supporting automatic modeling initiatives.

Threat modeling ontology framework

The goal of our ontology framework is to support the automation of threat modeling by improving the comparability and completeness of data from multiple sources. It considers the following types of data elements: (i) software products, (ii) operating systems, (iii) other types of applications and, (iv) data flows. The requirements for the reasoning functionality of the ontology framework are based on earlier studies (Farwick et al. 2011). The desired functionality is the standardization, classification, and grouping of the data elements, but not all operations are supported on all data elements.

Because every company has their own infrastructure and development environment, it is essential that the ontology is flexible and adaptable in order to work for any given context. Some organizations may have customized or even self-built tools that should be included in the threat modeling. Therefore, it is important that the ontology gives room for customization. Because our framework is rather broad it should capture most desired items, such that in case a user should want to modify parts in a way that for some reason does not comply with the framework, these additions could be added to the model manually. If this is not enough, the user could even aspire to add to the framework by developing a new branch to the ontology.

The ontology framework builds on ontology patterns as defined in Gangemi and Presutti (2009) and Falbo et al. (2013), which are created in different phases of ontology development as explained in Falbo et al. (2013). The patterns are provided to simplify partial, but also complete reuse of the framework. The ontology framework's development process starts with conceptual modeling and design phases, during which reusable ontology patterns (content patterns and reasoning patterns) are produced. The last phase is the implementation phase, in which the ontology patterns are implemented. The next two subsections explain the ontology patterns, and the third subsection gives an example of ontology implementation and knowledge acquisition. "Case study" section presents the use of the implemented ontology with a case study using multiple datasets. Figure 2 gives an overview of the three phases covered in this section.

Content patterns

Content patterns solve conceptual ontology design problems by addressing modeling issues for the domain classes and their properties. Content patterns can be reused as building blocks in the ontology design. The following subsections present the content patterns that are the building blocks of the ontology framework. The subsections partially follow the template from (Gangemi and Presutti 2009). We begin the description of the content patterns with a short summary of each pattern. Subsequently, a generic use case (Intent) is explained, followed by competency questions that each ontology must address. Next, a diagram is presented to show the elements in the pattern and their relationships, and further descriptions of the elements are provided. Finally, the process of creating specific patterns is briefly explained. Some of the patterns, such as *"Standardization of application software names"*,*"Classification of application software"*, and *"Grouping of application software names"* may initially appear to be similar. They are separated as individual patterns because this structure fits well with the structure of the SecuriCAD framework, which we use in our case study.

Standardization of operating system names

Summary Heterogeneous data sources can use different names to represent the same object (such as a system or software). The different representation of names complicates the merging of data from multiple sources. If two or more data sources use different naming conventions or formatting, it can be hard to distinguish whether two objects that are being referred to are the same or are different. For example, one data source might say that a computer node is running "Microsoft Windows Server 2003 Standard Edition," whereas another refer to the same operating system as "Microsoft Windows Server 2003 SE." Such a difference will require a conflict resolution during the merging process to decide which data source, if any, makes the correct claim. If the names had been standardized in advance, the conflict resolution would not be required. Thus, before the merging process, all names should be standardized to the same format and syntax.

Intent The intent is to represent standard operating system names of different platforms, and their variations, which can be used to standardize the operating system names obtained from multiple data sources for data merging.

Competency questions

1. What different names and name representations do common operating systems have?

Diagram The diagram of the pattern is shown in Fig. 3.

Elements

`OperatingSystemPlatform`: Represent the major platform of the software. Examples include Windows, Linux, and Unix. The reason for differentiating is the existence of different naming conventions.
`OperatingSystemName`: The standard version of the operating system name without a version.
`OperatingSystemVersion`: The different version an operating system can have.
`OperatingSystemVendor`: The distributor of the particular operating system software.
`executes`: A relation between `OperatingSystemPlatform` and `OperatingSystemName`
`enumeratedAs`: A relation between `OperatingSystemName` and `OperatingSystemVersion`
`createdBy`: A relation between `OperatingSystemName` and `OperatingSystemVendor`

Creation

1. Extract data from the data model patterns of IT data sources.

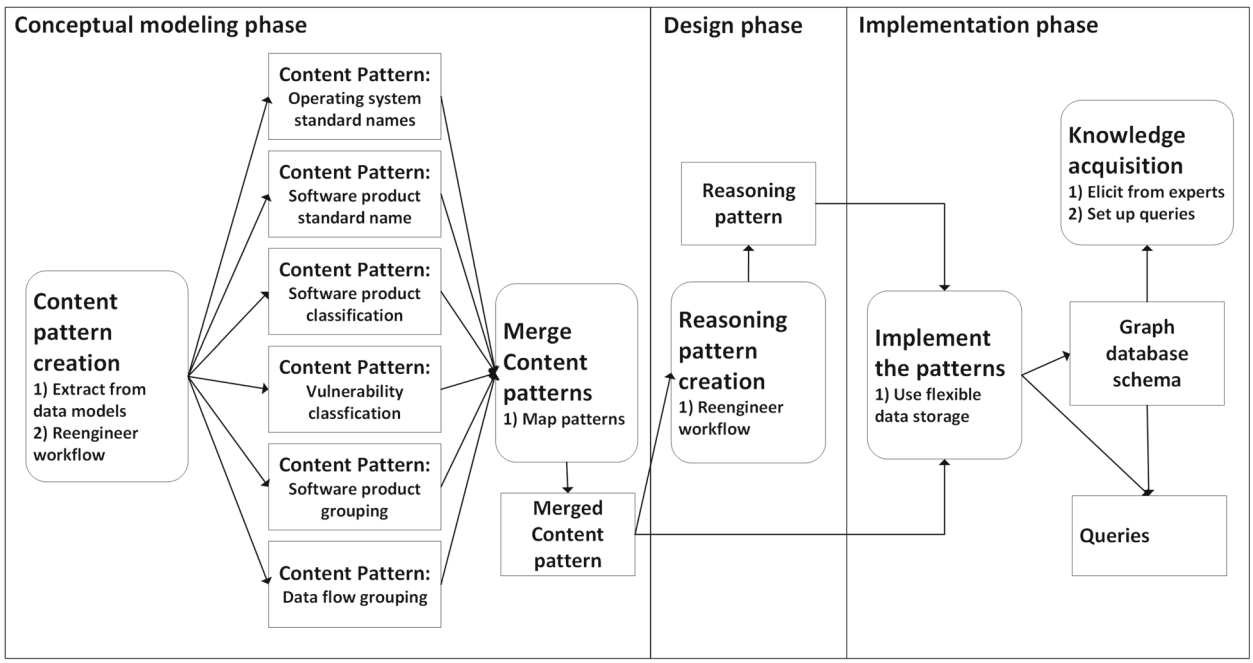

Fig. 2 Ontology framework life cycle from specification to implementation. Operations are represented with rounded blue rectangles and artifacts with normal rectangles

2. Reengineering from the automatic model creation framework workflow pattern.

Standardization of application software names

Summary The pattern of standardizing of application names is similar to that of standardizing operating system names ("Standardization of operating system names" section). The goal of this pattern is to standardize heterogeneous representations of software product (application) names from heterogeneous data sources. The standardization of application names needs a different approach than that of operating system names because the names for different software products can be created by different vendors.

An application is represented by a name and is usually accompanied by a version number. Depending on the platform, a vendor name can also be available. An application can be modular, meaning it can have several parts, thus there might be a main name and a sub-name describing some specific functionality of the application. The version number often consists of the major version, which represents a major release of the software, and a minor part which is increased incrementally with every update. An application name might also contain information about

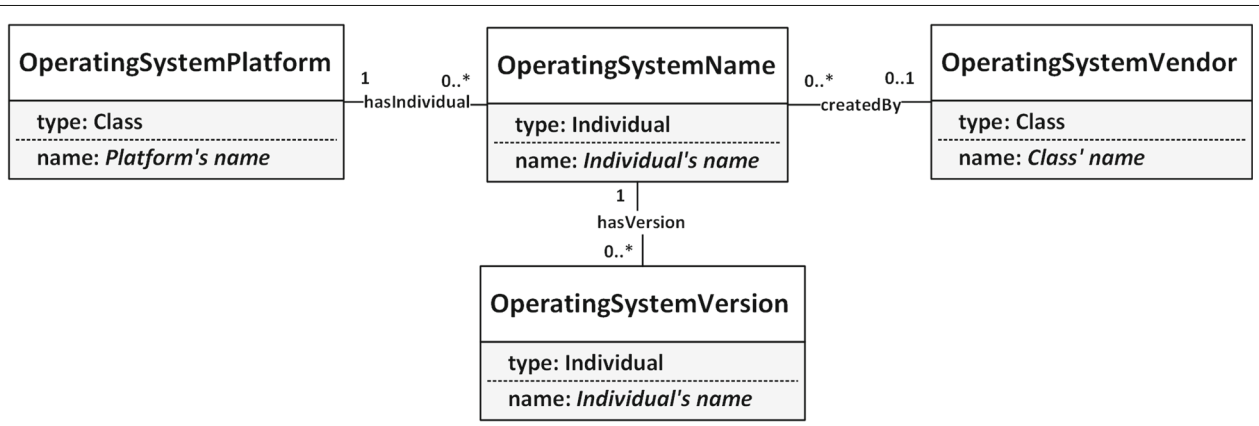

Fig. 3 Standardization of operating system names

major fixes, such as if a certain version of a service pack has been installed. All of this is represented in the pattern.

Intent The intent is to represent standard software product names for different platforms and their variations, which can be used to standardize the names obtained from multiple data sources when merging the data.

Competency questions

1. What different names, versions, and representations do common software products have?

Diagram The diagram of the pattern is shown in Fig. 4.

Elements

`Platform`: Represents the main type of operating system on which the software runs. Examples include Windows, Linux, Unix. Naming conventions for different platforms differ.
`ApplicationName`: The standard version of the software product name without a version.
`ApplicationMainVersion`: The major versions of a software product.
`ApplicationVersion`: The versions of a software product.
`ApplicationArchitecture`: The architectures of a software product. Examples include x86 or x64.
`ApplicationVendor`: The vendor of a particular software product.
`executes`: A relationship between `Platform` and `ApplicationName`
`enumeratedBy`: A relation between `ApplicationName` and `ApplicationMainVersion`
`has`: A relation between `ApplicationMainVersion` and `ApplicationVersion`
`executesOn`: A relation between `ApplicationVersion` and `ApplicationArchitecture`
`createdBy`: A relation between `ApplicationName` and `ApplicationVendor`

Creation

1. Extract data from the data model patterns of IT data sources and from public software repositories.
2. Reengineering from the automatic model creation framework workflow pattern.

Classification of application software

Summary System utilities and network scanning tools such as Nexpose and Nessus often only give names of application software installed on particular hosts. Using this kind of data to create models requires expert knowledge and manual supervision when classification of application software is required. If such knowledge and supervision is not available, as with automated modeling, other means are required to perform the classification.

Intent The intent is to classify application software according to the predefined set of classes such as client or server software.

Competency questions

1. In what application software classes are we interested?
2. To what class (from the identified classes) does a particular application software product belong?

Diagram The diagram of the pattern is shown in Fig. 5.

Elements

`Platform`: Represents the general type of operating system on which the application runs.
`ApplicationName`: Represents the application name of the software product.
`ApplicationClass`: The application type such as server, client, or data store.
`executes`: A relation between `Platform` and `ApplicationName`.
`isKind`: A relation between `ApplicationName` and `ApplicationClass`.

Creation

1. Extraction from IT data sources and from the data models of modeling tools.

Classification of vulnerabilities

Summary Software vulnerabilities can be used to model and assess threats against particular systems. When a modeling tool does not support the inclusion of all the software vulnerabilities in the model, software vulnerabilities must be grouped according to some criteria.

A widely used source for vulnerability data is the National Vulnerability Database (NVD).[1] NVD uses a community-developed Common Weakness Enumeration list (CWE)[2] to categorize NVD software vulnerability entries into general weakness types. However, there are more than 900 CWE weaknesses, and therefore, a small number of meaningful groups might be required. A logical way to group software vulnerabilities is according to the attack methods that can be used on them, and that is the purpose of this pattern.

[1] https://nvd.nist.gov Accessed 2020-07-27
[2] http://cwe.mitre.org/ Accessed 2020-07-27

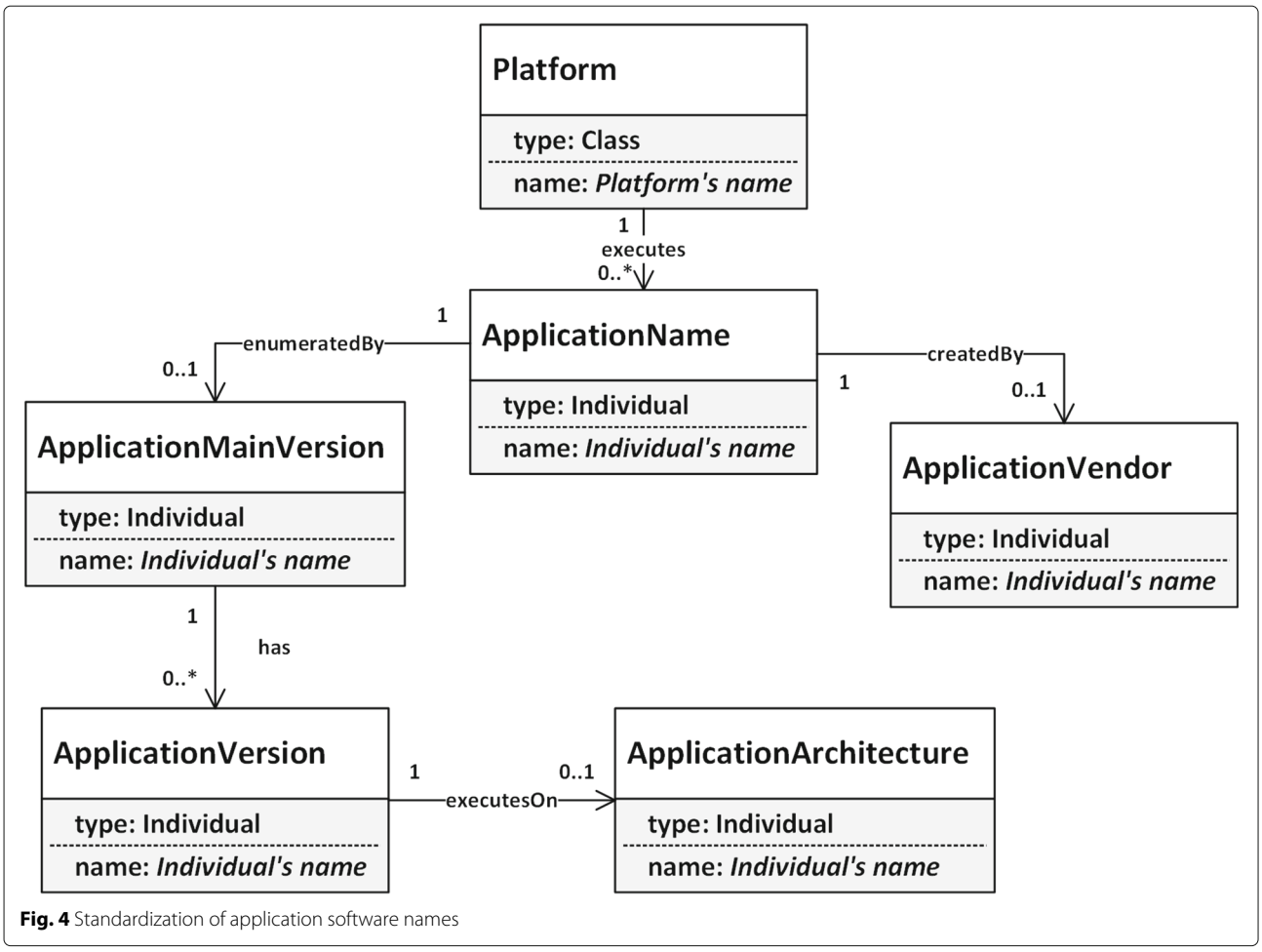

Fig. 4 Standardization of application software names

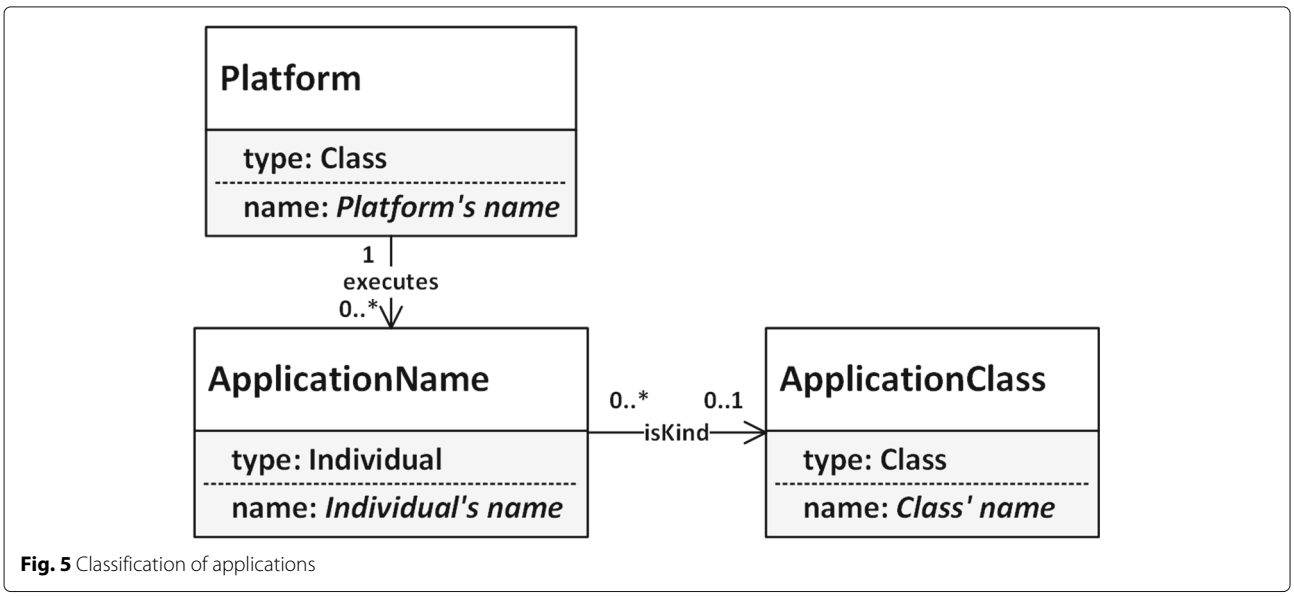

Fig. 5 Classification of applications

Intent The intent is to enable grouping software vulnerabilities from the NVD into meaningful groups, such as attack method based groups.

Competency questions

1. What attack method from a known list of attack methods can be used against a software product?

Diagram The diagram of the pattern is shown in Fig. 6.

Elements

`AttackMethod`: Represents a general type of attack method that can be used against a specific software vulnerability.
`CWE`: A community driven software weakness category entry. Most NVD entries (CVEs) have one assigned to them and this relationship in NVD is used to assign CVEs to AttackMethods during knowledge acquisition.
`CVE`: An NVD vulnerability database entry denoting a single software vulnerability of one or more software products.
`CPE`: An entry from the Common Platform Enumeration (CPE)[3], which is a dictionary of common software applications and hardware names. Many software products have a unique CPE entry.
`relatesTo`: A relation between `AttackMethod` and `CWE`.
`groups`: A relation between `AttackMethod` and `CVE`.
`has`: A relation between `CVE` and `CWE`.
`impacts`: A relation between `CVE` and `CPE`.

Creation

1. Extraction from the data models of modeling tools and from public software vulnerability and weakness repositories.

Grouping of application software names

Summary Each threat model assumes a certain level of granularity. In automated modeling, many data sources are used that can have different granularities compared to the modeling language. Though it is impossible to increase the granularity if additional information is not available, it is possible to reduce the granularity even with no additional information available, and a reduction is often what we want to do. The purpose of grouping functionality is to facilitate the reduction of granularity for application software modeling.

[3] https://cpe.mitre.org/cpe Accessed 2020-07-27

Intent The intent of the pattern is to facilitate the grouping of application software into meaningful groups, similar to how multiple Microsoft Office products are grouped into a single Office Suite.

Competency questions

1. Can a particular application software be grouped?
2. If a particular application software can be grouped, what is the name of the group?

Diagram The diagram of the pattern is shown in Fig. 7.

Elements

`Platform`: The general type of operating system on which the application runs.
`ApplicationName`: The name of the application in question.
`ApplicationGroup`: The group assignment of a particular software application.
`executes`: A relation between `Platform` and `ApplicationName`.
`groupedInto`: A relation between `ApplicationName` and `ApplicationGroup`.

Creation

1. Extraction from IT data sources and from the data models of modeling tools.

Grouping of data flows

Summary The focus of the data flow grouping pattern is similar in form to that of the application software pattern. However, the data flow grouping pattern adapts data granularity to a modeling language level for application level data flow modeling, whereas the application software pattern adapts the description of software.

An application level data flow as defined in the pattern is a flow of similar network packets between two network hosts. The pattern captures the numbers and names of the relevant application protocols that should be modeled.

Intent The intent of the pattern is to facilitate the grouping of time series network packet captures into time independent application data flows between two or more hosts.

Competency questions

1. Is there an application data flow between network hosts?

Diagram The diagram of the pattern is shown in Fig. 8.

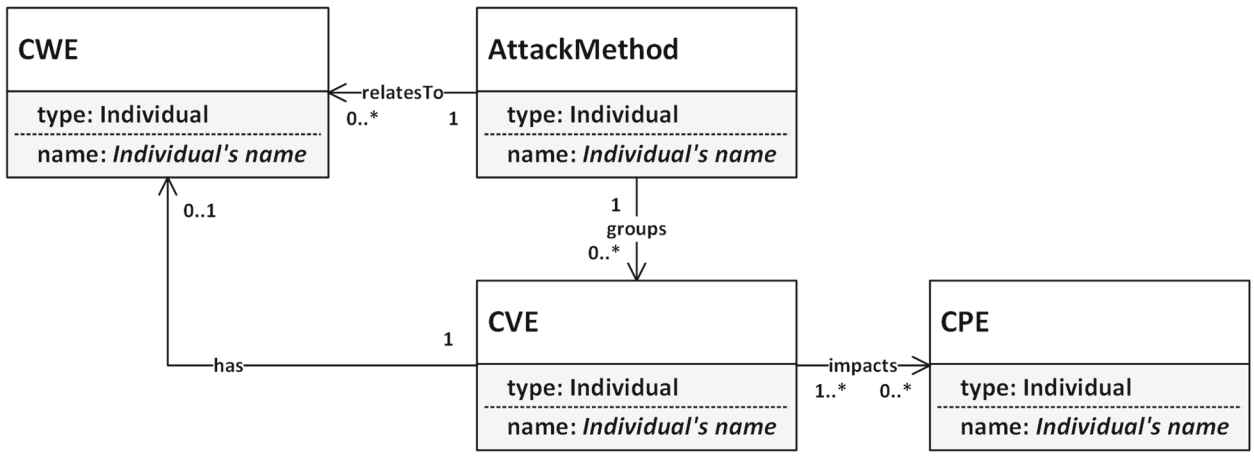

Fig. 6 Classification of vulnerabilities

Elements

`HostToHostProtocol`: The type of the host to host network communication protocol. Two values are possible, either TCP or UDP.
`PortNumber`: The port number of the server in a one way communication.
`ApplicationProtocol`: The name of the application protocol that is linked to a particular port.
`MaxNonDynamicPort`: The last non dynamic port as defined by IANA,[4] so that the larger port numbers can be safely ignored.
`includes`: Relation between `HostToHostProtocol` and `PortNumber`.
`translatesTo`: Relation between `PortNumber` and `ApplicationProtocol`.
`limitedBy`: Relation between `HostToHostProtocol` and `MaxNonDynamicPort`.

Creation

1. Extraction from IT data sources and from the data models of modeling tools.

Reasoning pattern

The purpose of the reasoning pattern is to address design problems related to queries on the ontology. The reasoning patterns described here can be considered as a functional extension of the ontology. The following paragraphs explain the reasoning required for each part of the ontology framework. Each reasoning pattern description starts with an explanation regarding the input to a query, then the output, and finally a short description of how the reasoning was performed.

Standardization of operating system names The reasoning pattern for standardization of operating system names uses stored acquired knowledge, exclusion information and is described below.

- Input: Software product name, software product version, vendor name.
- Output: Standardized product name, standardized version, vendor name.
- Processing steps i) Exclude general words from the name such as 'Kernel' depending on the amount of words in the name. ii) Separate the name, product name and version using known names, or the type of characters in the name. iii) Use a word based match (word order unimportant) of known ontology product name values. iv) Return the database query result.

Standardization of application software names The standardization of application names is achieved by applying platform specific word exclusion and word matching strategies. Here, the platforms are seen as general classes of operating systems, such as Linux or Windows. The need for a separate strategy results from different naming conventions being used on each platform. The queries use stored, acquired knowledge, and is described below.

- Input: Application software name, software version, vendor name, platform.
- Output: Standardized application name, standardized version, vendor name.
- Processing steps i) Based on platform, exclude architecture, distribution and language version information (like 'ENU' for Microsoft) from product name. ii) Discard all the characters except the ones that describe the platform based products best. iii) Use a word based match (word order unimportant) of

[4] https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml Accessed 2020-07-27

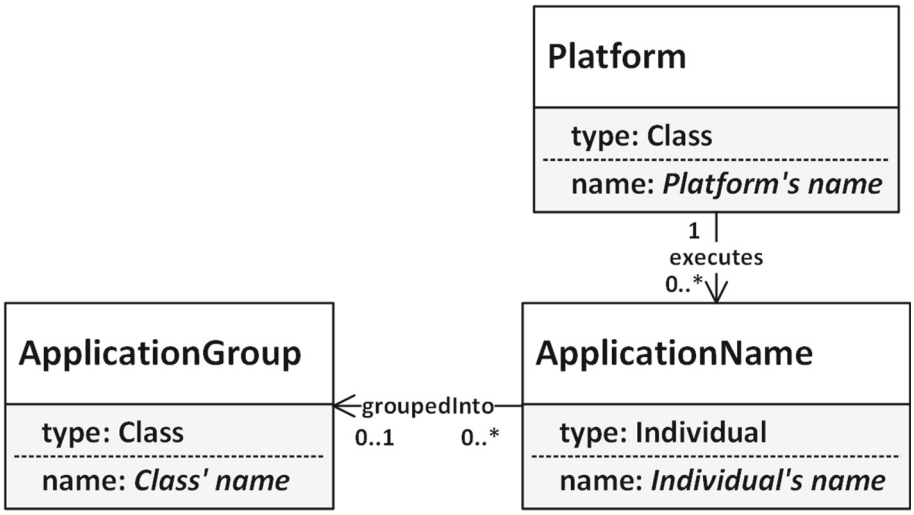

Fig. 7 Grouping of software names

known ontology product name values. iv) Return the database query result.

Classification of application software The classification of software is achieved by using pre-stored software classification information and matching input standardized software product names to known software product names and related classes. Version and vendor information are completely ignored as they are not relevant to determining the class of the application.

- Input: Standardized application software name.
- Output: Application class, such as Server.
- Processing steps i) Use a word based match of known ontology product name values. ii) Return the database query result.

Classification of vulnerabilities The role of the vulnerability classification functionality is to classify software vulnerabilities according to more general attack type classes. The authors see the main data source for vulnerability data as NVD, and the unique weaknesses as being identifiable by CWE ids. The idea of vulnerability classification then is to substitute CWEs with a smaller number of predetermined attack categories, and to use these categories to classify the software vulnerabilities (such as CVE) that have a linked CWE value in the NVD.

- Input: Software vulnerability ID.
- Output: Attack type class like XSS.
- Processing steps i) Use a word based match of CVE number to find corresponding attack type. ii) Return the database query result.

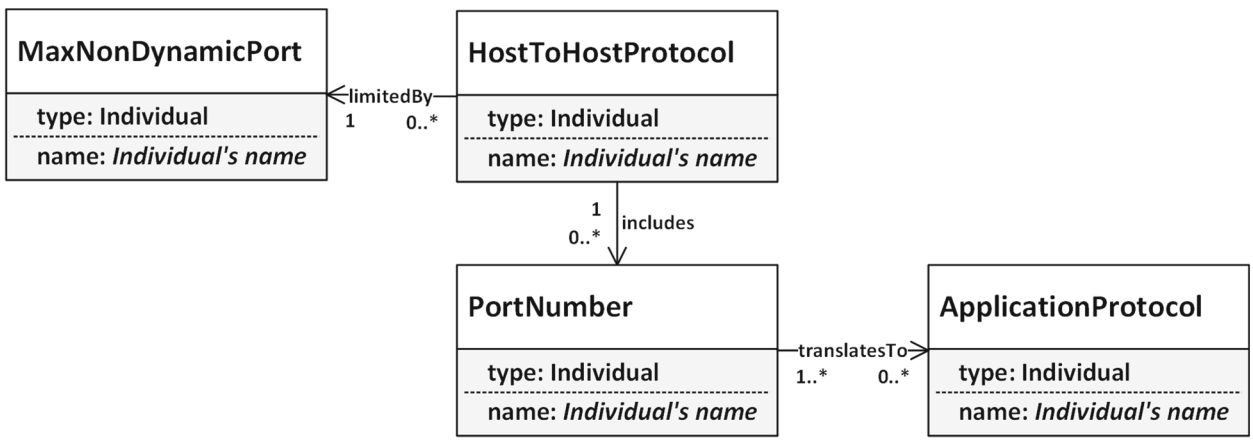

Fig. 8 Grouping of data flows

Grouping of application software names The method for grouping software titles involves identifying if a particular application software is part of a software suite using acquired knowledge. Due to each platform having a different naming convention, platform specific strategies must be applied. For Linux applications the first part of the name usually represents the primary part of the software, and for grouping purposes, the rest can be discarded. Windows applications need to have been defined in the ontology.

- Input: Standardized application name.
- Output: Group name.
- Processing steps i) Use a word based match to find predefined group for a particular application software name. ii) Return the database query result.

Grouping of data flows The grouping of data flows means ignoring the time aspect of the data flows and abstracting multiple network package exchanges into unique, but fewer traffic flows. The ontology in this case is needed to identify the application data flows using port numbers. Dynamic port ranges are excluded from the results.

- Input: Source address, destination address, destination port, transport protocol.
- Output: Source address, destination address, application protocol name.
- Processing steps i) Transport protocol is matched. ii) A query to find an application protocol name corresponding to a port number. iii) Return the database query result.

Ontology framework implementation and knowledge acquisition

This subsection describes the implementation of the ontology content and the reasoning patterns introduced in "Content patterns" and "Reasoning pattern" sections as an ontology. As part of the ontology creation, knowledge needs to be acquired to populate the ontology. The knowledge acquisition is also covered in this section.

The ontology content patterns are implemented as a graph data model in a hybrid NoSQL database. For this study, the database engine ArangoDB[5] is used. Because ontology is often seen as a finite set of unambiguously identifiable classes and their relationships, a graph data model is a logical choice. In addition, NoSQL databases provide the flexibility and scalability that the traditional relational database systems fail to provide. The data structure of the ontology, which is based on the described content patterns, is shown in Fig. 9. The ontology is made up of classes and related individuals that are manifestations of the classes. The data model consists of nodes that represent the classes and individuals and are connected to each other with named edges. The ontology reasoning patterns are implemented as database (AQL) queries and Python functions. The Python functions are implemented in a way so that the reasoning patterns described in "Reasoning pattern" section can be used for automated modeling to improve the data quality of incoming data.

The ontology framework is plugged into the modeling automation process that has been implemented as part of an earlier work (Vlja et al. 2019). However, the ontology framework is based on a set of functions that could be invoked from other systems and the automation modeling process is not needed for using the ontology framework. The integration between the two as implemented in this study is shown in Fig. 10. As shown on the figure, the ontology framework functions are invoked from the modeling automation process adapters. There is a separate adapter for each type of data source. The idea of the adapter is to process files from a particular type of data source that show the state of the data source at a particular moment. The goal of the ontology framework is to enrich this data using the functions shown in the figure. Once the data has been enriched it is used as input to the modeling automation process.

Knowledge acquisition is required to fill the ontology with knowledge. The role of the knowledge acquisition is to collect data that corresponds to the data model and can be applied as knowledge. According to (Pinto and Martins 2004), there are many possibilities for knowledge acquisition such as elicitation techniques, inductive techniques, or using relevant data sources. Maedche and Staab (2001b) mention extraction of ontology knowledge from web documents using resource processing components. In our case, knowledge acquisition is conducted separately for each content pattern using methods coded in Python.

For the first two content patterns, standard operating system names and application names, data is acquired from two sources, internet software repositories and the lab environment described in "Case study" section. In both cases the data is collected for two platforms, Windows and Linux, and platform specific strategies are applied to standardize the names. The main reason for choosing the Windows and Linux platforms is the availability of data.

The operating system names are collected using various data sources – network scanners and system software utilities like Windows Powershell and yum. The names are parsed with the help of platform based translation lists, which capture acronyms and their fully written definitions. For example 'SE' translates to 'Standard Edition' for the Windows platform. In the case of Linux, we need to exclude unnecessary information. For example, depending

[5] https://www.arangodb.com Accessed 2020-07-27

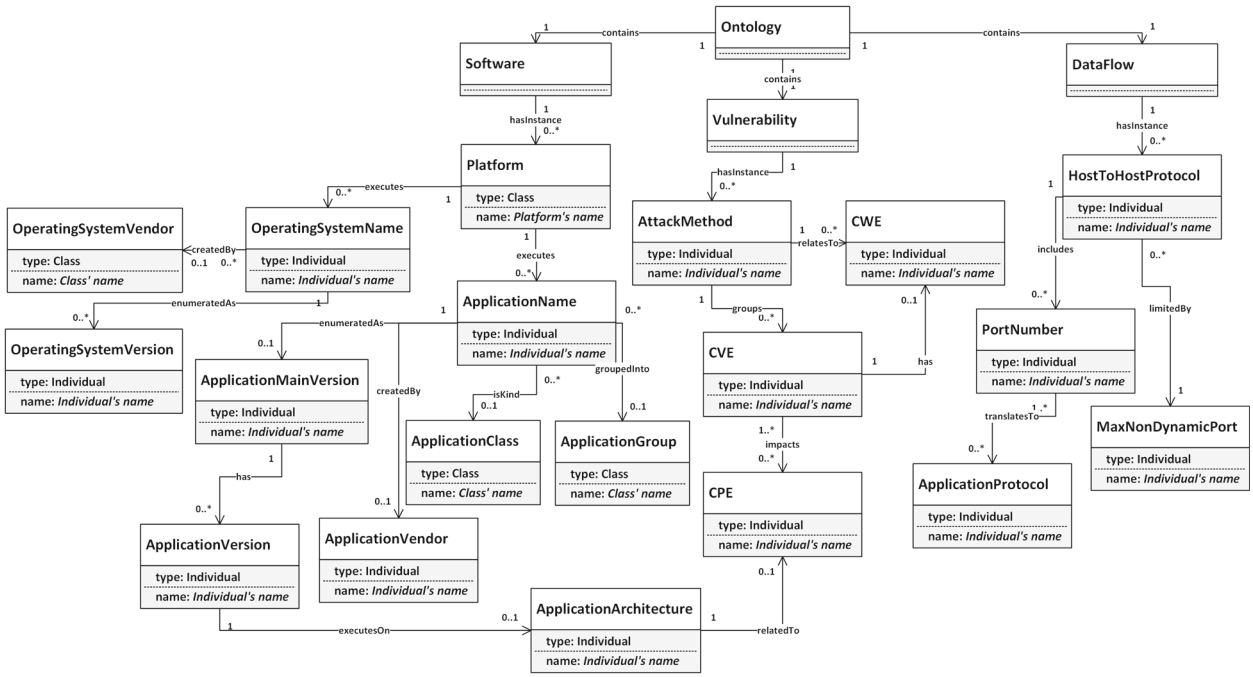

Fig. 9 Database schema created from merged content patterns

on the amount of words in the name, we might want to remove 'Linux' or 'Linux Kernel' from the full operating system names. During the name capture, the version information is stored separately from the product name to enable abstraction to the product name only.

An application software element is represented by a unique name and a version number. Depending on the platform, the ontology might also contain its vendor and platform information. The strategy to add software names to the ontology uses platform specific regular expression

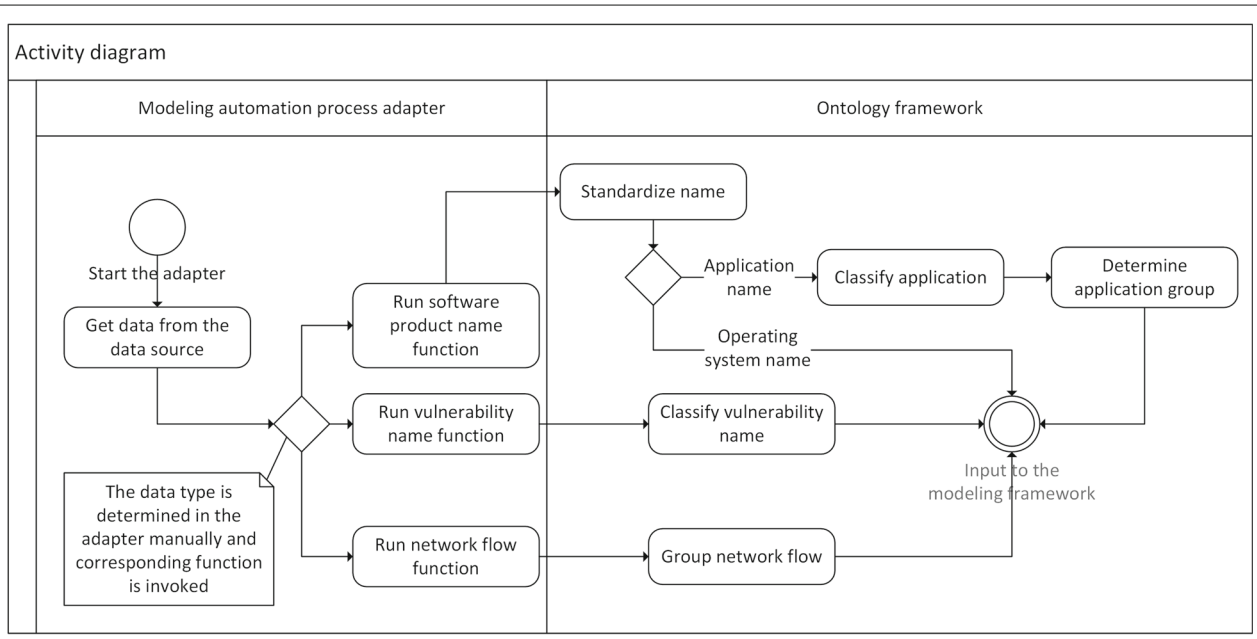

Fig. 10 Implementation activity diagram

based extraction of the name and exclusion lists. For both platforms, architecture specific information is excluded from the application names, and for Windows, language version specific acronyms like 'ENU' are also excluded when possible. The strategy to add application software version numbers is also based on platform specific exclusion lists. A Linux application version number is often made up of the package version number which shows the version of the application, such as '4.5', and a Linux distribution release number, such as '3.fc26', which shows the distribution specific changes to the code. As this level of detail is not required by the content pattern (and for our purposes), the release information is removed. Once the product and operating systems names get stored in the ontology database, they become the standard names.

The third content pattern is about classification of application software. First, a set of classes needs to be defined. The classes are taken from the modeling tool SECURICAD used in our case study (see "Case study" section). The applications are classified according to the following categories: client, server, data store, firewall, scanner, operating system internals. Here again a distinction between operating systems needs to be made and different strategies need to be used. In case of applications that can be installed on the Windows platform, no good source of classification data could be identified. Thus, we used our expertise to classify the collected application names.

There are more alternatives for classifying Linux software. Web sites like Linux Packages Search[6] contain detailed information about Linux packages, including short and long descriptions. Moreover, the packages on the Linux Packages Search web site are organized according to distributions and sometimes the type of the application is also available. In our solution we download a description of each package name learned from our systems, and then sort them into the predefined classes according to known application names and combinations of keywords. Internal operating system packages and libraries get marked as well so that they can be excluded at will.

The vulnerability content pattern supports grouping vulnerabilities based on attack methods. The vulnerabilities can be classified into four attack groups:

- Command injection.
- Remote file injection.
- SQL injection.
- Cross site scripting (XSS).

As CVE identification numbers are used in many popular vulnerability scanners like Nessus and Nexpose, the classification content pattern is based on the National Vulnerability Database (NVD) standard that uses CVE identification numbers by default. In the NVD dataset, the vulnerabilities (with CVE id) have been classified into various weakness types (represented by a unique CWE id). To map the CVE ids to the specific attack method groups, first the CWE need to be matched to these groups. The Open Web Application Security Project (OWASP) regularly publishes a list of the top 10 application security risks.[7] MITRE on the other hand has created CWE OWASP Top 10 views, where CWEs are linked to the different attacks in the OWASP top 10 list.[8] For the purpose of categorization, we choose four attack methods from the list and create 4 groups of CWE ids. These groups are then compared to all the CVE ids and the CWE ids linked to the CVE ids. That way, the software vulnerabilities that belong to one of the four groups can be stored in the database and later compared to incoming CVE ids to get the group name.

The four chosen categories capture a large part of the relevant vulnerabilities and fit well in our ontology framework because they are supported by the SECURICAD modeling tool. Together, they allow the modeler to create an accurate representation of the threats an organization may face while maintaining a comprehensive structure. More information regarding the implementation and validation of the ontology framework and of the SECURICAD tool can be found in "Case study" section.

The application name grouping content patterns help to adapt the granularity level of data. We apply platform specific strategies because the names for different platforms follow different formats. Windows specific names use multiple parts that are separated by spaces. For Windows application names it is difficult if not impossible to say how many first parts of the name define a group, meaning that a manual group definition is needed. Linux package name parts are separated by hyphens and follow a more unified format. Usually, the first part of the name is representative for the entire group, such as perl or mysql. Therefore the first part is always used as the group name.

Similar to the previous content pattern, the role of the data flow pattern is to adapt the granularity of the data to the required level. The idea of the data flow grouping is to exclude the time aspect and only keep certain application protocols. In network traffic, port numbers are used to distinguish different types of traffic between a source and a destination. Thus we need a list of known port numbers that link to application protocols. Such a source is provided by Internet Assigned Numbers Authority (IANA).[9] However, because the list is general and application configurations might differ between organizations, it makes

[6] https://pkgs.org Accessed 2020-07-27

[7] https://www.owasp.org/index.php/Top_10-2017_Top_10

[8] https://cwe.mitre.org/data/definitions/928.html Accessed

[9] https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml Accessed 2020-07-27

sense to keep a customized version of this list in each separate environment of the ontology.

Case study

This section presents a case study with three different enterprise system setups. The datasets from the three setups are used to create cyber threat models of IT architectures using the modeling tool SECURICAD[10] (Ekstedt et al. 2015). The metamodel (model structure) of SECURICAD has a central role in the implementation of the ontology framework, by determining the data needs and constraints for the implemented ontology. The implementation of the ontology framework is described in "Ontology framework implementation and knowledge acquisition" section.

Three cases are described as they complement each other. The first case (SCADA lab) contains the most comprehensive dataset and demonstrates the functionality of the ontology framework in full. The second case (Water utility field devices) shows the usefulness of grouping data flows for higher level modeling when the protocols are similar and the differences mainly lie in the payload of the network packages. The third case (CRATE lab) demonstrates how data flow grouping makes sense even if the exchanged data are diverse (multiple different protocols) and exchanges take place between a large number of hosts. The nature of the data from the third setup facilitates the extraction of the operating system names for some hosts using the fingerprinting tool, p0f. This is not possible with the second dataset owing to the homogeneous nature of its captured traffic.

SECURICAD,[11] the modeling tool used in all three cases, is a tool for threat modeling and risk assessment. SECURICAD is designed to be used for evaluation of cybersecurity of systems-of-systems-level architectures. Attack graphs are automatically derived from the modeled architecture, and time-to-compromise values are calculated to provide estimations on the likelihood of various attacks towards the architecture. The attack graph follows the logic of a Bayesian network. The security properties of different assets have a relatively high abstraction level, meaning that individual vulnerabilities are less considered than general expectations of a system deploying the latest patches. SECURICAD's metamodel describes 14 different IT architecture asset types, such as network, host, software, dataflow, and access control.

Datasets

The following sections detail the threedata sets used in this study and their origins.

SCADA lab

Supervisory Control and Data Acquisition (SCADA) systems are widely used in industrial control systems such as manufacturing, electricity supply, and fresh and waste water utilities. SCADA systems play an important role in the automation of industrial processes, as they enable an operator to oversee a complex process that may be geographically distributed and to take action as necessary. SCADA systems do not include the real-time control logic in machinery, but rather use sensors and actuators on top of these in order to provide an interface to the human operator. As SCADA systems thus offer this link between previously standalone systems and the rest of the world, SCADA system security has a received a lot of attention in the past few decades (Korman et al. 2017; Knapp and Langill 2014).

Our SCADA lab is virtualized with ten servers and runs using ABB SCADA software. The network has been segmented into SCADA, Office, Management, and Substation zones. A firewall restricts traffic between the zones. Some of the virtual machines run Windows Server 2012 software, whereas other machines run various Linux distributions, including Red Hat Enterprise Linux 7.

The lab data used to test the ontology framework include configuration files from operating systems and network devices and results from network scans conducted using Nmap,[12] Nessus,[13] and Nexpose[14] scanners. Because of the variety of data available, all functions of the ontology framework can be tested. This includes standardization of operating system and software names, classification of applications and vulnerabilities, and also grouping of applications and data flows.

Water utility field devices

The data from the second case study are obtained from a water utility. Similarly to the first case, the data are collected from an environment where industrial control systems control industrial processes. Unlike the first case, the data are obtained from a real geographically distributed production environment.

The dataset contains network traffic between the water utility's central control system and its field devices. The traffic is in the form of TCPDump files and consists mostly of control messages that are MODBUS over TCP/IP being exchanged between AC800M (controller) and AC500 (programmable logic controller) devices and that have been captured using a sniffer. There is a firewall between the field devices and the control devices that stops all the non-control traffic.

[10] https://www.foreseeti.com/securicad/ Accessed 2020-07-27
[11] https://www.foreseeti.com/products Accessed 2020-07-27
[12] https://nmap.org/download.html Accessed 2020-07-27
[13] https://www.tenable.com/products/nessus/nessus-professional Accessed 2020-07-27
[14] https://www.rapid7.com/products/nexpose Accessed 2020-07-27

The dataset contains detailed information regarding the source, destination, and content of the network package in a time sequence. As the data in question is MODBUS over TCP/IP traffic, the source of the traffic are the controllers and destinations are the programmable logic controllers. The content of the packages are commands that write to and read from the programmable logic controllers' memory values.

CRATE lab

The Swedish Defence Research Agency is running a Cyber Range And Training Environment (CRATE).[15] This environment supports deploying thousands of virtual machines and is equipped with host based traffic generators.

Occasionally, cyber security related training exercises are conducted in CRATE. The third dataset pertains to one of those exercises and contains general descriptions of the virtual machines and network traffic captured at various points in time between more than 50 hosts. Various activities between the hosts have been captured, including scanning and hacking. The network traffic is in the form of TCPDump files.

The types of actions available for this dataset are standardization of operating system names and grouping of data flows.

Application of the framework

The goal of the application of the ontology framework is to improve semantic accuracy and the match of granularity for SECURICAD (explained at the beginning of the "Datasets" section) model elements. The ontology framework is applied in an automated modeling process (from an earlier work Blind) and the application of the framework is described according to the reasoning patterns for all three available datasets.

The data are captured as files and the files are processed using data source specific adapters. The data sources are shown in Table 1. During the automated modeling process, the data source specific adapters transform data into a common format and write the transformed data to a back end database. Seven different adapters are used For the SCADA lab case study, one (Wireshark) is used for the Water utility, and two (p0f, Wireshark) are used for CRATE.

The adapters in our study are run twice to store the data in a common format in two different back end databases for comparison. During the first run of the adapters, no ontology functionality is used. The second run is done with the ontology functionality enabled. The following paragraphs explain the differences between those runs for the implemented reasoning patterns.

[15] https://www.foi.se/en/foi/resources/crate---cyber-range-and-training-environment.html Accessed 2020-07-27

Table 1 Collected data and the different methods used for collecting the data. For each data source a separate data source specific adapter is created to parse the data

Source	Method	Data description
Windows 2012 Server	Powershell cmdlet	Installed software
Red Hat Enterprise Linux 7	yum list	Installed software
Nessus	Scanning	Vulnerability data, operating system names
Nexpose	Scanning	Vulnerability data, operating system names
Nmap	Scanning	Service names, operating system names
p0f	Passive capture	Data flows, operating system names
Wireshark	Passive capture	Data flows

Standardization of operating system names During the standardization of operating system names, syntactic and linguistic differences of operating system names are addressed. Following a common format makes the names from multiple sources comparable with each other. The SCADA lab dataset comprises 88 operating system names from four different data sources. Among these 88 names, 38 are distinct. After applying ontology framework, the number of distinct names is reduced to 33.

In the CRATE dataset, only one data source provided operating system names, thus standardization of the names did not achieve anything for that data set. In total, 13 operating system names are captured, of which five are distinct.

Standardization of application software names During the standardization of application software names, syntactic and linguistic differences of application software names are addressed. Following a common format makes the names from multiple sources comparable with each other. The SCADA lab setup comprises 2097 application software names, out of which, 1076 are distinct, captured by five sources. After the ontology framework's application, 1041 distinct names remain.

Classification of application software The purpose of application software classification is to recognize different predefined types of application software in the automatically collected data and to add that knowledge to the final model. In the SCADA dataset there is a large number of application software lacking type information (1442). After applying the ontology framework, only 585 undefined names remain. The rest of the application software titles are sorted into either server (initially 578, afterwards

587), client (initially 77, afterwards 396), data store (initially 0, afterwards 16), or firewall (initially 0, afterwards 4) types. The ontology framework also sorts 440 titles into the excludable group, which we have designated for operating system internal software packages and libraries. Removing them from the model helps to match the level of abstraction of the model better.

Classification of vulnerabilities The SECURICAD model only supports four general attack based vulnerability categories (command injection, remote file inclusion, SQL injection, and cross-site scripting). Thus, if vulnerability scanners provide more detailed information this needs to be adapted to the model. Only the SCADA lab has vulnerability data available from 2 data sources. Initially, 1190 vulnerabilities are identified, out of which 504 are distinct. Only one category matches the identified vulnerabilities and is included in the model.

Grouping of application software names The purpose of grouping application software names is to reduce the amount of model elements through abstraction to meaningful software groups. For example, Microsoft Word 2013, Excel 2013 and Outlook 2013 might be grouped as Microsoft Office 2013 suite. For Linux software, often the tools provide detailed package names for each application software. This information is also not useful for SECURICAD modeling. Application software names can be grouped only for the SCADA lab dataset. Five different data sources provide software names (2097 total), out of which 1076 are distinct. After grouping the number of software application is reduced to 1401.

Grouping of data flows The goal of grouping data flows is to exclude the temporal aspect of communication, package payloads, and to avoid duplicated model elements to improve understandability of the model. The data flow grouping is done based on source, destination network addresses (IPv4) and the protocol name. Only application protocol data is included in the final model.

All three datasets provide data flow information. The SCADA lab dataset includes 76,802 unique packet exchanges recorded between various hosts, only 91 of which remain after data flow grouping. The Water utility dataset contains 25 739 461 different network package exchanges of similar nature (mostly MODBUS over TCP/IP) with different package payloads. Thus, after the package content data and temporal information has been excluded, only 102 unique data flows between 27 unique source and 32 unique destination network addresses (IPv4) remain. The network communications captured with the CRATE lab dataset are more diverse than those captured with the Water utility dataset. Among 284 237 data flows between 51 hosts, we are interested in only 111.

Discussion and conclusion

We set out to investigate if an ontology can be used to improve an automated threat modeling process. The goal of the work was to design an ontology framework to improve semantic accuracy and granularity match for threat models.

Our framework was designed using conceptual modeling. The models captured the reasoning functionality and the life cycle of the ontology framework. Both of these designs abstracted away from the implementation details. To implement the ontology framework, we needed models that could capture more details. Content patterns were useful for implementing the ontology and helped capturing details for the implementation, while also allowing for modularity of the design. Each content pattern supported some functionality, and if new functionality is needed for the framework, the patterns can be modified with this functionality.

The conceptual models and the ontology patterns are the basis for creating the ontology framework and automating the threat modeling process. The main evaluation criteria as stated in the literature was the usefulness of the models. The framework allows us to standardize and classify software while increasing the abstraction to ignore unnecessary details when appropriate.

The patterns proposed in the paper can be considered generic for a certain threat model and data source and can be reused whenever this kind of support is needed. However, all models leave out some properties of the systems that they study and are thus, from an absolutist sense, incomplete. Whether those excluded properties are needed to fulfill a particular task has to be determined by the users.

The results of applying the ontology framework on the three datasets show that it can improve the threat modeling process with domain knowledge where automation is applied. The results show that the semantic accuracy of the models can be improved using standardization and classification. The ontology based approach also helps to address the granularity mismatch problem, by abstracting away from unneeded information. In mismatch cases where information is missing, such as types of application software, the level of detail can be increased. The benefits are evident with small heterogeneous datasets, but the approach is even more useful in a large IT environment. Since the goal of the study was to improve the quality of the data with context that only a human modeler would be able to add, the outcome was evaluated by the authors who have extensive experience in modeling and data science. After several iterations that included various improvements, the results were as expected and the artifacts were deemed to meet the goals of the research.

Practical implications of the research are that missing information can lower the precision of decision support

models. This is especially true for models of decision support tools which can do quantitative analysis and simulations, such as SECURICAD. Our ontology framework was able to classify numerous application software titles and reduce the number of unclassified ones. As each class behaves differently in the model, this has a direct impact to the analysis results.

As discussed in "Related work" section, ontologies have existed for some time in various shapes and forms. Our current market analysis found that few ontologies target threat modeling, even fewer target our analyzed branch of automating threat modeling, and no ontology found by the authors delivers a comprehensive description of the construction and implementation of the framework. This demonstrates the uniqueness of the ontology presented in this article. Furthermore, no found ontology towards threat modeling treats the implementation and use towards industrial control systems or systems of critical infrastructure. With that said it is beneficial for the community if more people study and develop ontologies since they must be kept up to date in order to best facilitate the population of relevant data to threat models. Others could contribute to our model since it is based on an open source development and thus allows for efficient updates and even the potential of being forked to new sub-domains.

Although significant manual input is required for setting up the ontology and its knowledge acquisition process, there are no good alternatives as far as we know. Spending the time and effort set up an ontology is a one-time endeavor, and the ontology can then be used to improve an unlimited number of models with similar requirements. The alternative to using an ontology requires manually improving and enhancing each new or updated model, which becomes more time and effort intensive through repetition.

This takes us to the important observation that the quality of the models depends on the quality of the information in the ontology. The information in the ontology needs to be quality assured and updated regularly to maintain the quality of the models it is used to create. Automating the acquisition and updating of the knowledge in the ontology would significantly reduce the amount of work required to set up and maintain the ontology. Several machine learning algorithms were tested when developing this framework with the aim of speeding up knowledge acquisition and to ensure data quality, but the small size of available knowledge data limited their usefulness. Figuring out how to automate knowledge acquisition with this limitation is part of future research.

Acknowledgements
The authors would like to thank Göran Ericsson at the Swedish National Grid for the valuable discussions on the topics covered in this paper.

Authors' contributions
Margus Välja overall work on ontologies and ontology patterns, practical implementation of the framework in the case studies, and authoring. Fredrik Heiding contributing work regarding threat modeling, adapting the framework to fit the automation of threat models, and authoring. Ulrik Franke contributing work on ontologies, the implementation of ontologies in threat models, and authoring. Robert Lagerström: Contributing work on ontologies, the implementation of ontologies in threat models, and authoring. All author(s) read and approved the final manuscript.

Funding
This work has received funding from the European Unions H2020 research and innovation programme under the Grant Agreement No. 832907, Swedish Governmental Agency for Innovation Systems (Vinnova), the Swedish Energy Agency, SweGRIDS, and STandUP for Energy.

Availability of data and materials
The datasets used for the case studies and the script used for automating the population of data are available at the *project webpage* or from the *Dropbox folder*.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]KTH Royal Institute of Technology, 100 44 Stockholm, Sweden. [2]RISE Research Institutes of Sweden, 164 40 Kista, Sweden.

Received: 13 May 2020 Accepted: 2 September 2020
Published online: 01 October 2020

References

Aier S, Buckl S, Franke U, Gleichauf B, Johnson P, Närman P, Schweda CM, Ullberg J (2009) A survival analysis of application life spans based on enterprise architecture models. In: Mendling J, Rinderle-Ma S, Esswein W (eds). Enterprise modelling and information systems architectures : Proceedings of the 3rd international workshop on enterprise modelling and information systems architectures. vol. LNI P-152. Gesellschaft für Informatik, Bonn. pp 141–154

Aier S, Gleichauf B, Saat J, Winter R (2009) Complexity levels of representing dynamics in ea planning. In: Albani A, Barjis J, Dietz JLG (eds). Advances in Enterprise Engineering III. Springer, Berlin. pp 55–69

Akhawe D, Barth A, Lam PE, Mitchell J, Song D (2010) Towards a formal foundation of web security. In: 23rd IEEE Computer Security Foundations Symposium. IEEE, Edinburgh. pp 290–304

Antunes G, Bakhshandeh M, Mayer R, Borbinha J, Caetano A (2014) Using ontologies for enterprise architecture integration and analysis. Compl Syst Informa Model Q 1(1):1–23. https://doi.org/10.7250/csimq.2014-1.01 https://www.ingentaconnect.com/content/doaj/22559922/2014/00000001/00000001/art00001

Antunes G, Borbinha J, Caetano A (2016) An application of semantic techniques to the analysis of enterprise architecture models. In: 49th Hawaii International Conference on System Sciences (HICSS). IEEE, Honolulu. pp 4536–4545. https://doi.org/10.1109/HICSS.2016.564

Antunes C, Caetano A, Borbinha J (2014) Enterprise architecture model analysis using description logics. In: IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations. IEEE, Ulm. pp 237–244. https://doi.org/10.1109/EDOCW.2014.43

Axelsson J (2015) A systematic mapping of the research literature on system-of-systems engineering. In: 10th System of Systems Engineering Conference (SoSE). IEEE, San Antonio. pp 18–23. https://doi.org/10.1109/SYSOSE.2015.7151918

Barankova II, Mikhailova UV, Afanaseva MV (2020) Minimizing information security risks based on security threat modeling. J Phys Conf Ser 1441:012031

Berger BJ, Sohr K, Koschke R (2013) Extracting and analyzing the implemented security architecture of business applications. In: 17th European conference on software maintenance and reengineering. IEEE, Genova. pp 285–294. https://doi.org/10.1109/CSMR.2013.37

Caldarola EG, Picariello A, Rinaldi AM (2015) An approach to ontology integration for ontology reuse in knowledge based digital ecosystems. In: Proceedings of the 7th international conference on management of computational and collective intelligence in digital ecosystems. MEDES '15. ACM, New York. pp 1–8. https://doi.org/2857218.2857219
Cardenas AA, Roosta T, Sastry S (2009) Rethinking security properties, threat models, and the design space in sensor networks: A case study in scada systems. Ad Hoc Netw 7(8):1434–1447. https://doi.org/10.1016/j.adhoc.2009.04.012 http://www.sciencedirect.com/science/article/pii/S1570870509000468, privacy and Security in Wireless Sensor and Ad Hoc Networks
Catak FO, Yilmaz M, Gul E (2019) Sensor based cyber attack detections in critical infrastructures using deep learning algorithms. Comput Sci 20:213. https://doi.org/10.7494/csci.2019.20.2.3191
Cesare S. d, Foy G, Lycett M (2016) 4d-setl. In: Proceedings of the 18th international conference on enterprise information systems. ICEIS. SCITEPRESS - Science and Technology Publications, Lda, Portugal. pp 127–134. https://doi.org/10.5220/0005822501270134
Chen Y, Boehm B, Sheppard L (2007) Value driven security threat modeling based on attack path analysis. In: 40th Annual Hawaii International Conference on System Sciences (HICSS'07). p 280a. https://doi.org/10.1109/HICSS.2007.601
Chhaya B, Jafer S, Proietti P, Marco BD (2019) An ontology for threat modeling and simulation of small unmanned aerial vehicles. In: 9th International Defense and Homeland Security Simulation Workshop, DHSS 2019. Springer, Cham. pp 23–28
Dhillon D (2011) Developer-driven threat modeling: Lessons learned in the trenches. IEEE Secur Priv 9(4):41–47
Ekelhart A, Fenz S, Klemen MD, Weippl ER (2006) Security ontology: Simulating threats to corporate assets, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). International Conference on Information Systems Security 4332:249–259. LNCS cited By :20
Ekstedt M, Johnson P, Lagerström R, Gorton D, Nydrén J, Shahzad K (2015) Securi cad by foreseeti: A cad tool for enterprise cyber security management. In: IEEE 19th International Enterprise Distributed Object Computing Workshop. IEEE, Adelaide. pp 152–155
Falbo RA, Guizzardi G, Gangemi A, Presutti V (2013) Ontology patterns: clarifying concepts and terminology. In: Proceedings of the 4th international conference on ontology and semantic web patterns-volume 1188. CEUR-WS. org, Aachen. pp 14–26
Farwick M, Agreiter B, Breu R, Ryll S, Voges K, Hanschke I (2011) Requirements for automated enterprise architecture model maintenance. In: 13th International Conference on Enterprise Information Systems (ICEIS). SciTePress - Science and and Technology Publications, Beijing
Farwick M, Breu R, Hauder M, Roth S, Matthes F (2013) Enterprise architecture documentation: Empirical analysis of information sources for automation. In: System sciences (HICSS) 2013 46th hawaii international conference on. IEEE, Wailea. pp 3868–3877
Florez H, Snchez M, Villalobos J (2014) iarchimate: A tool for managing imperfection in enterprise models. In: 2014 IEEE 18th international enterprise distributed object computing conference workshops and demonstrations. IEEE, Ulm. pp 201–210. https://doi.org/10.1109/EDOCW.2014.38
Gangemi A, Presutti V (2009) Ontology design patterns. In: Handbook on ontologies. Springer, Berlin. pp 221–243
Gong L, Tian Y (2020) Threat modeling for cyber range: an ontology-based approach. Lect Notes Electr Eng 517:1055–1062
Gruber TR (1995) Toward principles for the design of ontologies used for knowledge sharing? Int J Hum-Comput Stud 43(5):907–928. https://doi.org/10.1006/ijhc.1995.1081 http://www.sciencedirect.com/science/article/pii/S1071581985710816
Guizzardi G, Herre H, Wagner G (2003) On the general ontological foundations of conceptual modeling. In: Spaccapietra S, March ST, Kambayashi Y (eds). Conceptual Modeling — ER. Springer Berlin Heidelberg, Berlin, Heidelberg. pp 65–78
Guizzardi G, Wagner G, Guarino N, van Sinderen M (2004) An ontologically well-founded profile for uml conceptual models. In: Persson A, Stirna J (eds). Advanced Information Systems Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg. pp 112–126
Hinkelmann K, Gerber A, Karagiannis D, Thoenssen B, Van der Merwe A, Woitsch R (2016) A new paradigm for the continuous alignment of business and IT: Combining enterprise architecture modelling and enterprise ontology. Comput Ind 79:77–86
Jarrar M, Demey J, Meersman R (2003) On using conceptual data modeling for ontology engineering. Springer Berlin Heidelberg, Berlin, Heidelberg
Jiang R, Lu R, Wang Y, Luo J, Shen C, Shen X (2014) Energy-theft detection issues for advanced metering infrastructure in smart grid. Tsinghua Sci Technol 19(2):105–120. https://doi.org/10.1109/TST.2014.6787363
Johnson P, Lagerstrm R, Ekstedt M (2018) A meta language for threat modeling and attack simulations. In: Proceedings of the 13th International Conference on Availability, Reliability and Security - ARES 2018 the 13th International Conference. ACM Press, Hamburg. pp 1–8
Johnson P, Lagerström R, Ekstedt M, Österlind M (2014) It management with enterprise architecture. KTH, Stockholm
Johnson P, Lagerström R, Närman P, Simonsson M (2007) Extended influence diagrams for system quality analysis. J Softw 2(3):30–42
Knapp ED, Langill JT (2014) Industrial network security: securing critical infrastructure networks for smart grid, scada, and other industrial control systems. Syngress 2:41–84
Korman M, Välja M, Björkman G, Ekstedt M, Vernotte A, Lagerström R (2017) Analyzing the effectiveness of attack countermeasures in a scada system. In: Proceedings of the 2nd Workshop on Cyber-Physical Security and Resilience in Smart Grids. Association for Computing Machinery, New York. pp 73–78
Kornecki AJ, Janusz Z (2015) Threat modeling for aviation computer security. CrossTalk 28(6):21–27
Krumay B, Bernroider E, Walser R (2018) Evaluation of Cybersecurity Management Controls and Metrics of Critical Infrastructures: A Literature Review Considering the NIST Cybersecurity Framework: 23rd Nordic Conference, NordSec 2018, Oslo, Norway, November 28-30, 2018. Proceedings 11252:369–384
Lagerström R, Johnson P, Ekstedt M (2010) Architecture analysis of enterprise systems modifiability: a metamodel for software change cost estimation. Softw Qual J 18(4):437–468
Lankhorst MM, van Buuren R, van Leeuwen D, Jonkers H, ter Doest H (2004) Enterprise architecture modelling – the issue of integration. Adv Eng Inform 18(4):205–216
Luh R, Schrittwieser S, Marschalek S (2016) Taon: An ontology-based approach to mitigating targeted attacks. In: ACM International conference proceeding series. Association for Computing Machinery, New York. pp 303–312. cited By :2
Maedche A, Staab S (2001) Ontology learning for the semantic web. IEEE Intell Syst 16(2):72–79. https://doi.org/10.1109/5254.920602
Maedche A, Staab S (2001) Ontology learning for the semantic web. IEEE Intell Syst 16(2):72–79
Maedche A, Staab S (2004) Ontology learning. Springer Berlin Heidelberg, Berlin, Heidelberg
Marback A, Do H, He K, Kondamarri S, Xu D (2013) A threat model-based approach to security testing. Softw Pract Experience 43(2):241–258. cited By :28
Marksteiner S, Ramler R, Sochor H (2019) Integrating threat modeling and automated test case generation into industrialized software security testing. In: ACM International Conference Proceeding Series. ACM Press, New York
Moral-Garca S, Moral-Rubio S, Rosado DG, Fernndez EB, Fernndez-Medina E (2014) Enterprise security pattern: A new type of security pattern. Secur Commun Netw 7(11):1670–1690. cited By :7
Pan S, Morris T, Adhikari U (2015) Developing a hybrid intrusion detection system using data mining for power systems. IEEE Trans Smart Grid 6:3104–3113. https://doi.org/10.1109/TSG.2015.2409775
Patil P, Pawar S (2012) Remote agent based automated framework for threat modelling, vulnerability testing of soa solutions and web services. In: World Congress on Internet Security (WorldCIS-2012). IEEE, Guelph. pp 127–131
Pei D, Zhang L, Massey D (2004) A framework for resilient internet routing protocols. IEEE Network 18(2):5–12. cited By :17
Pinto HS, Martins JP (2004) Ontologies: How can they be built? Knowl Inf Syst 6(4):441–464. https://doi.org/10.1007/s10115-003-0138-1
Pittl B, Fill HG, Honegger G (2017) Enabling risk-aware enterprise modeling using semantic annotations and visual rules. In: European Conference on Information Systems (ECIS), International. AIS, Guimarães
Rahm E, Do HH (2000) Data cleaning: Problems and current approaches. IEEE Data Eng Bull 23(4):3–13

Roth S, Hauder M, Farwick M, Breu R, Matthes F (2013) Enterprise architecture documentation: Current practices and future directions. In: Wirtschaftsinformatik Proceedings. AIS, Leipzig

Satnam Singh, Tu H, Allanach J, Areta J, Willett P, Krishna Pattipati (2004) Modeling threats. IEEE Potentials 23(3):18–21

Soffer P, Hadar I (2007) Applying ontology-based rules to conceptual modeling: a reflection on modeling decision making. Eur J Inf Syst 16(5):599–611. https://doi.org/10.1057/palgrave.ejis.3000683

Song F, Zacharewicz G, Chen D (2013) An ontology-driven framework towards building enterprise semantic information layer. Adv Eng Inform 27(1):38–50. https://doi.org/https://doi.org/10.1016/j.aei.2012.11.003 http://www.sciencedirect.com/science/article/pii/S1474034612001048, modeling, Extraction, and Transformation of Semantics in Computer Aided Engineering

Steven J (2010) Threat modeling - perhaps it's time. IEEE Secur Priv 8(3):83–86. https://doi.org/10.1109/MSP.2010.110

Torr P (2005) Demystifying the threat modeling process. IEEE Secur Priv 3(5):66–70. https://doi.org/10.1109/MSP.2005.119

Vasilecas O, Bugaite D, Trinkunas J (2006) On approach for enterprise ontology transformation into conceptual model. In: International Conference on Computer Systems and Technologies, CompSysTech. vol. 6. Association for Computing Machinery, New York

Vlja M, Lagerstrm R, Franke U, Ericsson G (2019) A framework for automatic it architecture modeling: Applying truth discovery. Complex Syst Inform Model Q 20:20–56

Xiong W, Lagerstrm R (2019) Threat modeling a systematic literature review. Comput Secur 84:53–69. cited By :5

Xu D, Nygard KE (2006) Threat-driven modeling and verification of secure software using aspect-oriented petri nets. IEEE Trans Softw Eng 32(4):265–278. cited By :112

Xu D, Tu M, Sanford M, Thomas L, Woodraska D, Xu W (2012) Automated security test generation with formal threat models. IEEE Trans Dependable Secure Comput 9(4):526–540. https://doi.org/10.1109/TDSC.2012.24

Publisher's Note