Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis

Concept for Security-Aware Modeling and Deployment of NFV Topologies Using TOSCA

Julian Sudendorf

Course of Study:	Software Engineering
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	Karoline Saatkamp, M.Sc.
Commenced:	April 9, 2018
Completed:	October 9, 2018

Abstract

Internet-based services, like cloud applications, increasingly become the target of cyber attacks. These attacks can range from data breaches of personal information to loss of data or severe financial damages. As a result, cybersecurity is a top priority for providers and users of these services. The networking layer plays a critical role in achieving security. Traditionally network functions that secure communications (for example firewalls or traffic encryption) are dedicated hardware appliances. Network function virtualization (NFV) is an emerging network architecture concept that utilizes virtualization to execute software implementations of network functions on standard IT infrastructure. Virtual Network Functions (VNFs) therefore become virtual software components that are usable in conjunction with conventional cloud application components.

The Topology and Orchestration Specification for Cloud Applications (TOSCA) is an OASIS standard to describe and manage cloud applications. A recent addition to the standard explicitly targets NFV based topologies. However, the standard does not make any assumptions on potential security problems and how to achieve enhanced security.

This thesis proposes a TOSCA based modeling concept to establish a connection between security threats of application topologies and VNFs that can mitigate these threats. The industry standard practice of threat modeling using the STRIDE method is employed to assess threats in application topologies. Based on present threats and available VNFs, automated recommendations can be made which VNFs should be used to enhance the security of cloud applications. A prototypical implementation in the context of Eclipse Winery, a modeling tool for TOSCA definitions, is used to validate the approach.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

List of Abbreviations

AWS	Amazon Web Services. 21
CAPEC	Common Attack Pattern Enumeration and Classification. 33
CAPEX	Capital Expenses. 17
COTS	customer off-the-shelf. 19
CP	Connection Point. 28
CPE	Customer Premise Equipment. 19
dA	Deployment Artifact. 25
DoS	Denial of Service. 32
ETSI	European Telecommunications Standards Institute. 19
GCP	Google Cloud Platform. 21
iA	Implementation Artifact. 25
IDS	Intrusion Detection System. 20
IoT	Internet of Things. 17
ISG	Industry Specification Group. 19
JSON	JavaScript Object Notation. 51
NAT	Network Address Translation. 19
NFV	Network Function Virtualization. 17
NIC	Network Interface Card. 27
NIST	National Institute of Standards and Technology. 31
OPEX	Operational Expenses. 17
OWASP	Open Web Application Security Project. 31
QName	qualified name. 41
SDN	Software-defined Networking. 22
SQL	Structured Query Language. 31
SSL	Secure Sockets Layer. 56
TOSCA	Topology and Orchestration Specification for Cloud Applications. 18

TSP Telecommunication Service Provider. 17

VDU Virtual Deployable Unit. 28

VIM Virtual Infrastructure Manager. 21

VLD Virtual Link Descriptor. 28

VM virtual machine. 23

VNF Virtual Network Function. 17

VPN Virtual Private Network. 17

WAN wide-area networking. 20

XML Extensible Markup Language. 23

YAML YAML Ain't Markup Language. 23

1 Introduction

In today's world, each day a vast number of new devices join the global connected world. As much as 125 billion devices, composed of computers and servers, mobile phones, Internet of Things (IoT) devices and other categories, are estimated to be connected to the internet by 2030 [IHS17]. Besides an exponentially growing number of connected devices, cybersecurity has become a top priority for users and providers of internet services [Gou15]. The WannaCry worm caused havoc in 2017 and infected over 200000 machines, encrypted their data and demanded a ransom [Ehr17]. Financial service provider Equifax suffered from a severe data leakage in 2017 due to an overseen security vulnerability in their application. This resulted in the theft of 230 million customer data sets including social security data [Ber17]. These two prominent examples are part of a rapidly expanding list of security incidents that caused millions of dollars in damages.

Telecommunication Service Providers (TSPs) and Network Operators have to host and manage large quantities of proprietary hardware appliances to be able to serve customers network and security needs. The ever-growing number of connected devices and their demand for new services poses multiple challenges to TSPs. These include (1) high Capital Expenses (CAPEX) for the acquisition of more specialized hardware, finding space to host the equipment (2) high Operational Expenses (OPEX) for setting it up and maintaining it [CCW+12] (3) account for optimal security of the provided services. Network Functions such as firewalls or Virtual Private Networks (VPNs), critically influence the security properties of a network. This in return impacts the applications that run inside a network and therefore the overall system security. The components of an application and their interactions with each other and external entities govern the security requirements the network has to take care of. This leads to the complex requirement to plan and instantiate security-related network functions based on the needs of an application that shall be protected. Due to the physical nature of hardware appliances, manual transport and reconfiguration are needed to make any topological change to an existing network, preventing agility regarding new services and service composition. The manual effort and the need to acquire physical hardware can be greatly reduced by realizing these network function purely in software and execute them on standard IT infrastructure instead of proprietary hardware. This concept is known as Network Function Virtualization (NFV).

NFV is an emerging networking architecture and has gained significant attention from academia and industry alike [MSG+15]. It builds upon the principle to segregate a network function's software from it's proprietary hardware by utilizing standard IT virtualization technology. This approach promises to significantly reduce both CAPEX and OPEX for providers while increasing time-to-market for new services. Additionally it provides the opportunity to instantiate and scale services dynamically where they are needed, for example at a network's edge or in the cloud. While offering these benefits, new challenges arise concerning modeling, deployment and management of these Virtual Network Functions (VNFs). The cloud computing domain faces similar challenges and closely relates to NFV in terms of utilizing virtualization technology.

The OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) standard enables the description of the deployment and management of cloud applications. An application's components, their relations to each other and the processes that manage them [OAS13] can be modeled. The generic nature of TOSCA allows to describe NFV topologies in a similar fashion as cloud applications. This was recognized by OASIS and a TOSCA profile for NFV was created to target the specific needs of NFV [OAS17]. Using TOSCA for NFV implies that network functions become entities that can be used in conjunction with regular application topologies. This results in a variety of new modeling opportunities. Security remains an objective that is not yet addressed by TOSCA, since the standard does neither make any assumptions on security [OAS13] nor provides any predefined entities to enhance security.

The objective of this thesis is to develop a concept for security-aware modeling and deployment of NFV topologies using TOSCA. It is necessary to investigate how TOSCA principles can be applied to NFV in general and what the resulting consequences are. Further, a security concept is developed and described to assist secure NFV utilization. The proposed concept is validated based on a prototypical implementation in the OpenTOSCA ecosystem, an open-source TOSCA implementation that enables modeling and deploying TOSCA based applications (see Section 2.3). The prototype is used to demonstrate how VNF can enhance application security. A demo use case is employed to validate the concept.

The remainder of this document is structured in the following way:

Chapter 2 - Theoretical Background and Fundamentals explains required background information and fundamentals this thesis is based on. It includes an introduction to NFV, the concepts of TOSCA, description of the OpenTOSCA ecosystem and a primer on security.

Chapter 3 - Related Work discusses the work related to this thesis regarding TOSCA based NFV management and security-aware modeling concepts.

Chapter 4 - Concept describes the approaches to enable security-awareness in modeling and deploying NFV topologies using TOSCA based on industry standard practices.

Chapter 5 - Implementation and Validation details the steps necessary to create a proof of concept for the proposed approach. This prototype is then used to validate the approach based on a demo use case.

Chapter 6 - Conclusion and Future Work summarizes the contributions of this thesis and gives a brief outlook on possible future work.

2 Theoretical Background and Fundamentals

The following chapter introduces terms and information required to understand the later presented concepts (see Chapter 4). First a definition of NFV is given (Section 2.1) followed by a brief introduction to the TOSCA standard (Section 2.2). The OpenTOSCA ecosystems and its components are explained afterwards (Section 2.3). An overview of security in context of the aforementioned topics concludes the chapter (Section 2.4).

2.1 Network Function Virtualization (NFV)

NFV is an emerging network architecture concept of how to design, deploy and manage Network Functions utilizing virtualization. Traditionally network functions such as firewalls or Network Address Translation (NAT) are realized by proprietary specialized hardware appliances and set up at specific locations in a network. These appliances need to be manually configured and connected by professionals to provide a service with certain characteristics. Changes to existing services or instantiation of new services result in high CAPEX and OPEX for providers. This is due to hardware that has to be acquired and the manual labor involved in setup and maintenance. The static nature of this approach prevents the TSP to react to demand in an agile fashion. The main concept of NFV is to decouple the functionality of a network function from the proprietary hardware and implement it in software. This software is then to be run on customer off-the-shelf (COTS) hardware, such as high-volume servers, memory and switches by leveraging virtualization technology. Thanks to virtualization many distinct network functions can be consolidated on the same hardware and dynamically moved to different locations [MSG+15]. These locations are determined by the presence of virtualization enabled hardware in the network. So a network function can be moved from a TSP's data center to the network's edge.

Figure 2.1 shows a traditional setup, composed of hardware network appliances on the left. The illustrated scenario shows what is required to establish a connection to private and public services from the perspective of a service provider and multiple customer sites. These sites can for example represent branch offices of a company that need connectivity. The traditional setup requires the same equipment on each customer premises, therefore aptly named Customer Premise Equipment (CPE). The right side of the illustration presents a possible NFV-based implementation. Here the necessary hardware CPE is reduced to a bare minimum and the remainder of the required network functions are virtualized on the service provider's premises. Both solutions provide the same functionality while the NFV-based approach can create more instances of functions when a new customer site is required or deprovision instances when the are no longer needed.

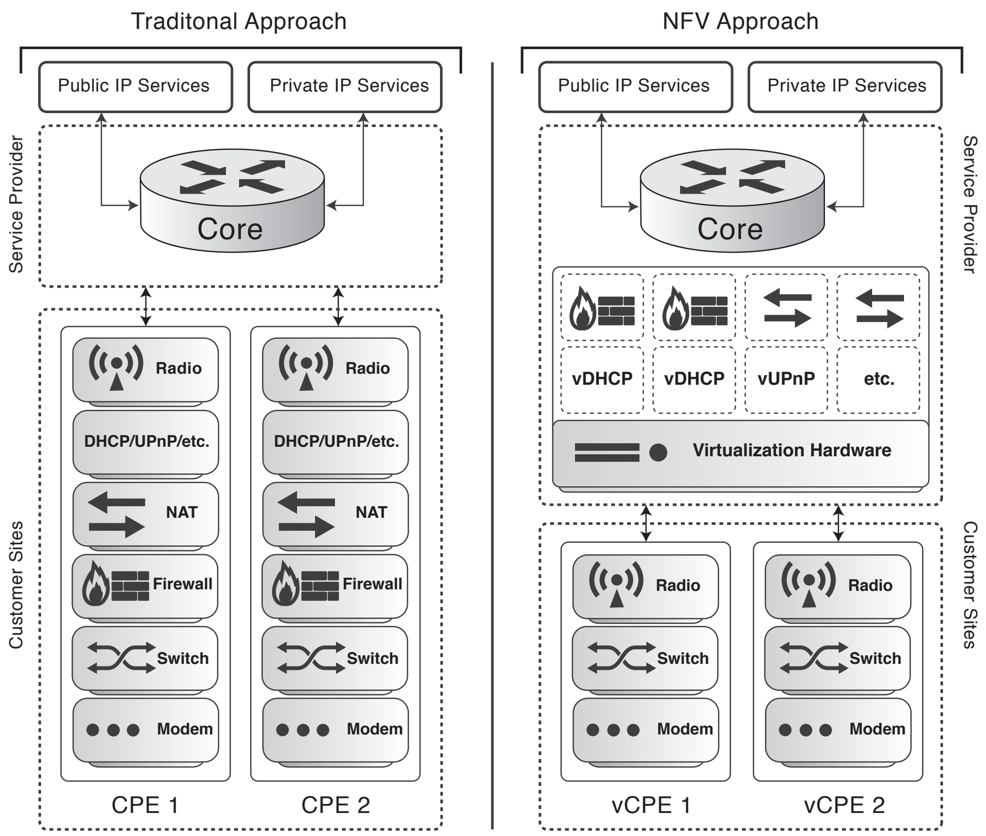


Figure 2.1: Comparison of traditional Customer Premise Equipment (CPE) (left) and one possible solution using a NFV approach (right), adopted from [MSG+15]

The concept of NFV was born in October 2012 when a number of the worlds leading TSPs authored a white paper calling for industrial and research action [CCW+12]. In November 2012 seven of these operators (AT&T, BT, Deutsche Telekom, Orange, Telecom Italia, Telefonica and Verizon) selected the European Telecommunications Standards Institute (ETSI) to be the home of the Industry Specification Group (ISG) for NFV (ETSI ISG NFV) [MSG+15]

2.1.1 Network Functions

Popular examples of network functions include load balancers, firewalls, Intrusion Detection System (IDS) or wide-area networking (WAN) optimization. A network function is a functional block within a network infrastructure that has well defined external interfaces and well-defined functional behavior [ETS14b]. As an example, when computers or applications in a private network need to communicate with an external network such as the internet multiple network functions are required. A router is used as a gateway from one network to the other. A firewall is used to control who can communicate with whom involving which protocols. Both functions need to expose external interfaces that need to be connected. The same applies for VNFs. A VNF is an implementation of a network function that is deployed on virtual resources such as a VM. A single VNF may be composed of multiple internal components. Hence it can be deployed over multiple VMs. In this case each VM hosts a single component of the VNF [ETS14a]. This is the result of decomposing network functions into smaller components that make up the same functionality when combined.

2.1.2 ETSI NFV reference architecture

ETSI specifies a NFV reference architecture[ETS14a] that is depicted in Figure 2.2. The following explanations are based on the standard document [ETS14a]. This architecture is used to define common components, the interfaces between them and to establish a common nomenclature. NFV Infrastructure (NFVI) represents diverse hardware resources that can be virtualized and exist in various locations. A location represents a so called point of presence in the network and is therefore called NFVI-PoP. The NFVI is where VNFs are executed. Element Management Systems (EMS) provide typical functionality to manage one or multiple VNFs. The right side of the diagram shows the NFV Management and Orchstration or in short MANO, which consists of the Orchestrator, one or more VNF Manager (VNFM) and the Virtual Infrastructure Manager (VIM). A VIM is the component that provides functionality that a VNF needs to interact with virtual compute, storage and network. A prominent example of a VIM is OpenStack[1], but alternatives like public cloud providers such as Amazon Web Services (AWS) or Google Cloud Platform (GCP) can also be considered. VNFM are used to manage the lifecycle operations such a instantiation or update of a VNF. The orchestrator coordinates VNFM and VIM to realize a Network Service. OSS/BSS, in the context of TSPs, stands for operations support system/business support system. In terms of NFV the BSS would get an order for a network service from a customer and the OSS fulfills the order and therefore connects to the NFV MANO. Descriptors for services, VNFs and infrastructure are stored and accessible by MANO. ETSI also proposes TOSCA (see Section 2.2) to be used as a descriptor for VNFs and services[ETS17].

[1]https://www.openstack.org/

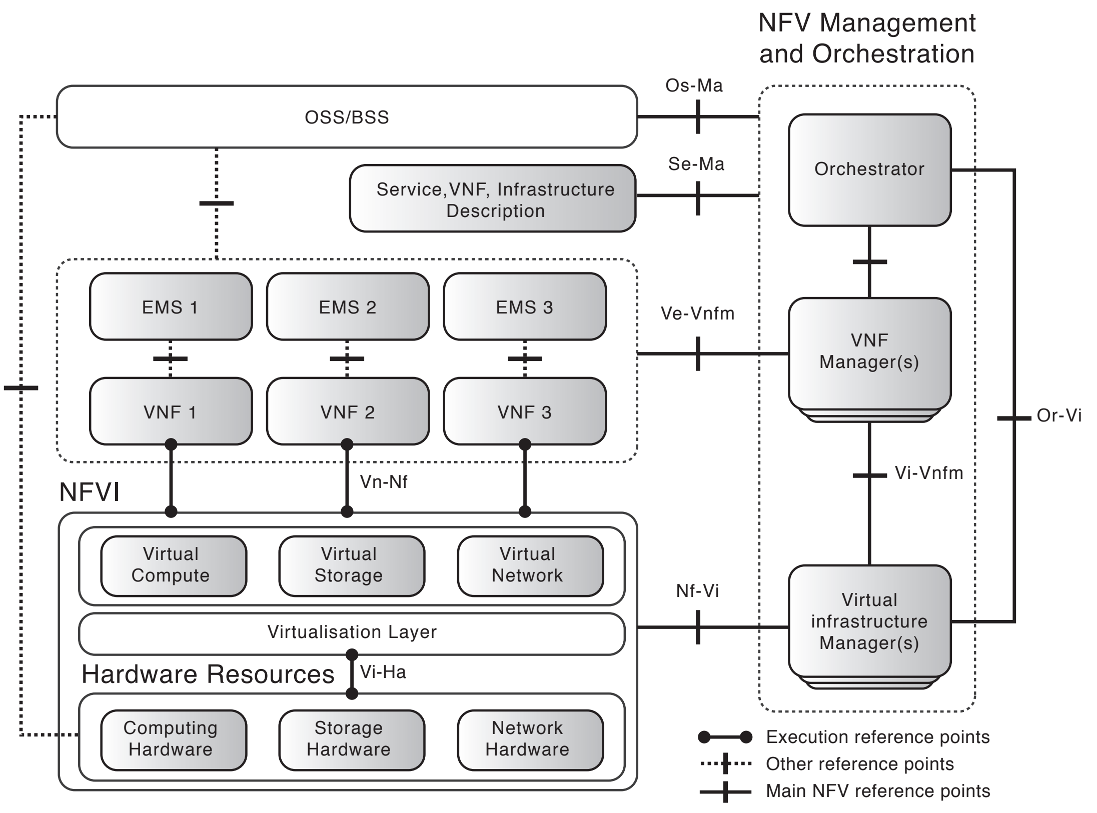


Figure 2.2: NFV Reference Architecture specified by ETSI, adopted from [ETS14a]

2.1.3 Distinction to Software-defined Networking

Software-defined Networking (SDN) is a term that is often mentioned when NFV is discussed and sometimes even incorrectly used interchangeably. The core principle of SDN is to decouple the the control plane from the forwarding plane [Sco17]. Traditionally a network is comprised of switches and routers and various other network appliances. As Figure 2.3 shows on the left, in a non-SDN network, each switch or router has a (often proprietary) integrated controller with an interface and needs to be addressed individually for changes. SDN aims to first separate the forwarding functionality from control and then centralize the control over multiple switches in a single controller that can address all switches. This is considered the control plane. Switches are therefore degraded to become simple packet forwarding devices that are programmable via an open interface (e.g. OpenFlow[Sco17]) by the controller. This is called the forwarding plane. An example of an SDN-based architecture is illustrated on the right side of Figure 2.3.

Virtualization comes into play since the controller can be realized in pure software and run on COTS hardware. The forwarding switches can be virtualized as well but it is not mandatory to do so[MSG+15]. While NFV and SDN both utilize virtualization and leverage automation the approaches are complementary[Sco17]. The desired outcome of

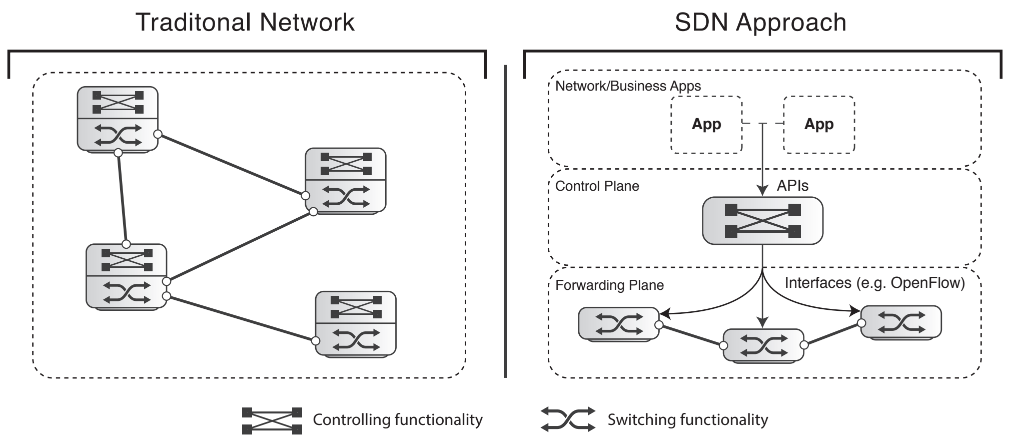


Figure 2.3: Traditional network architecture (left) compared to an SDN based approach (right), adapted from [MSG+15]

SDN is to deliver unified programmability to create virtual networks while NFV aims to virtualize network functions, deploy and scale them dynamically. Hence a combination of both approaches leads to advantages that lets each approach benefit from the other. One example of this is the ease of building service chains of VNFs by using SDN's network programmability concept [MSG+15].

2.2 Topology and Orchestration Specification for Cloud Applications (TOSCA)

TOSCA is an OASIS standard language which was introduced in 2013 [OAS13]. TOSCA defines a metamodel to describe the structure of composite cloud applications and the corresponding management tasks in a standardized and provider neutral way. Automated management, portability of applications and reusable application components are the three main goals of TOSCA [BBKL14a]. TOSCA 1.0 is based on Extensible Markup Language (XML) while more recent versions like the TOSCA SimpleProfile, that was introduced in 2016 , are based on YAML Ain't Markup Language (YAML) [OAS16]. The newer versions extend the standard and are backward compatible to TOSCA 1.0 [OAS16]. The XML-based notation of TOSCA 1.0 is therefore still valid and will be used throughout this thesis. An application modeled according to the TOSCA language is intended to be instantiated and managed by an TOSCA compatible orchestrator.

Figure 2.4 illustrates the topology of a cloud application based on WordPress and will be considered as a running example throughout this thesis. WordPress is a blogging application and used as a demo since it is a very commonly deployed scenario on the web [w3t18]. The topology consists of two Ubuntu 14.04 virtual machines (VMs). The left side represents the application code executed on a PHP runtime and served by an Apache web server. The right side describes the database-tier. The database VM hosts a database management system, namely MySQL and the actual database on top. Since WordPress

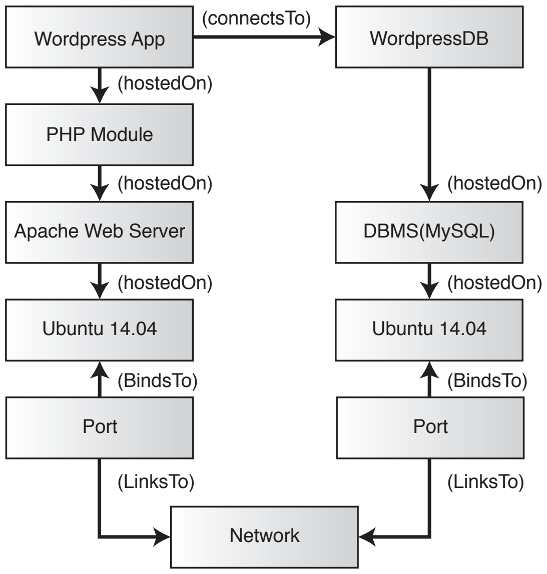


Figure 2.4: Running example of the application topology of a WordPress deployment

needs a database to store user information and blog posts, there is a relation between both VMs that indicates this requirement. Both VMs are bound to a separate port and linked to the same network.

2.2.1 TOSCA Entities and Packaging

The TOSCA language introduces a set of special entities that are used to establish the metamodel. Figure 2.5 illustrates and overview of these entities and their relations that make up TOSCA definitions.

Node Types define properties and lifecycle interfaces for a reusable entity. A Node Type can be derived from another Node Type. For example a VM Node Type can define properties for the name of the image that should be used and lifecycle interfaces for transferring a file to the VM.

Node Templates are instances of Node Types in the topology of a Service Template. A Node Template defines concrete values for the properties defined in the corresponding Node Type. In the case of the previously mentioned VM-Node Type example, Ubuntu 14.04 as image name.

Node Type Implementations define the actual implementations of Node Type lifecycle interfaces. To be able to transfer a file to a VM Node Type a software artifact needs to be present that can handle this specific functionality.

Relationship Types define the properties and lifecycle interfaces for relations that can be present between Node Templates. To indicate that one Node Type is hosted on another a *hostedOn* Relationship Type needs to be defined.

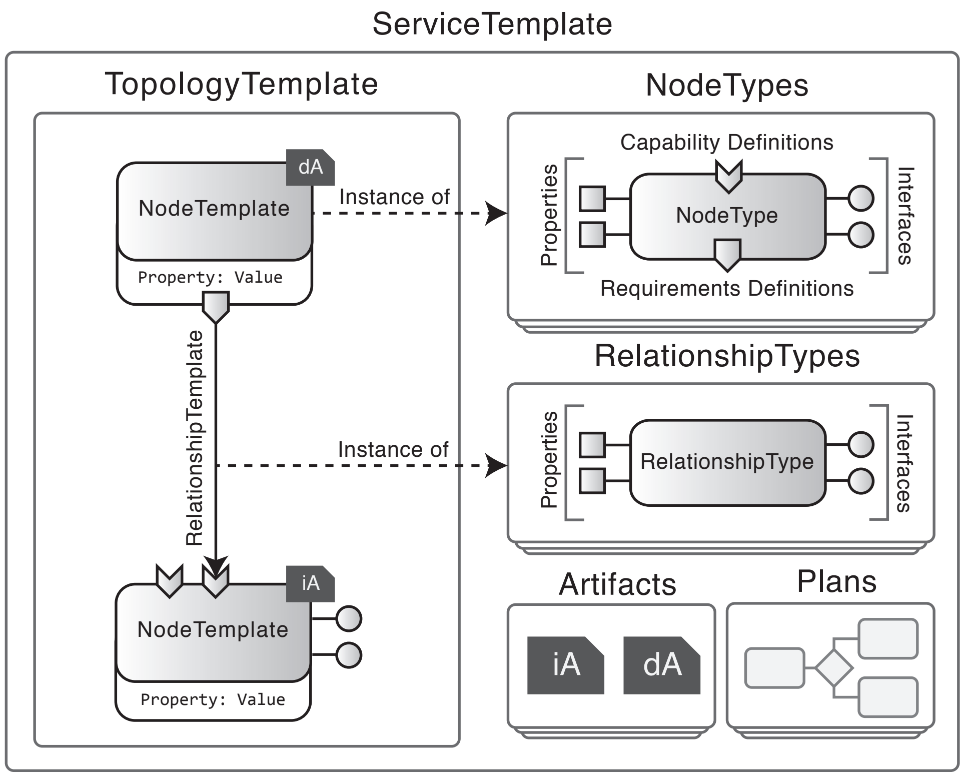


Figure 2.5: Hierarchy and entities of a TOSCA Service Template, adopted from [OAS13]

Relationship Templates are instances of Relationship Types in the topology of a Service Template. A Relationship Template manifests the actual relation between two Node Templates in the topology.

Requirement Types define requirements that a Node Type might have in regards to the relation to another Node Type. For example a software component like an Apache Web Server requires a compute node as a container to be hosted on in order to be executed.

Capability Types defines capabilities a Node Type can have in terms of possible relations. A capability of a Node Type can be seen as a feature it provides that satisfies the requirement of another Node Type. A VM Node Type has the capability to be the container, an Apache Web Server requires to be hosted on.

Artifact Types represent software artifacts as a reusable entity. Analogous to Node Types, properties can be defined.

Artifact Templates represent instances of a given Artifact Type. It is distinguished between *Implementation Artifacts (iAs)* and *Deployment Artifacts (dAs)*. *iAs* materialize the defined lifecycle interfaces of a Node Type or Relationship Type while *dAs* are files or resources that are needed by a specific Node Type. The actual WordPress files are a *dA* that needs to be transfered to a VM before the application can be run.

Policy Types define properties analogous to Node Types, Relationship Types and Artifact Types. Policy Types can be used express non-functional aspects regarding Quality of Service or placement for example.

Policy Templates are instances of Policy Types and attachable to Node Templates or Service Templates.

Service Templates describe the topology of an application as a graph where vertexes are Node Templates and edges are Relationship Templates. Additionally it contains the definition of all corresponding Types and Templates referenced in the topology.

In addition to the specified entities TOSCA allows to describe management plans and reference or embed them in a Service Template. To ensure portability of applications TOSCA introduces the *Cloud Service Archive*(CSAR) format. A CSAR is a zip encoded directory with a well-defined and extendable structure that contains all required definitions of an application [OAS13]. This results in a portable package that is fully self-contained and can be used in any TOSCA compatible environment [BBKL14a].

2.2.2 Substitution of Node Templates

TOSCA has a built-in concept to substitute Node Templates in the topology of a Service Template by the whole topology of another Service Template. A *Service Template* can define a *substitutable Node Type* attribute that indicates for which Node Type it can be used as a substitute. A Node Template is an instance of a Node Type and hence provides values for properties and defines the actual capabilities and requirements. The same is possible for Service Templates by defining so called *boundary definitions*. These boundary definitions allow to express properties, requirements and capabilities that are present inside the Service Template's topology. When a Service Template is substituted for a Node Template in another topology these boundary definitions are analyzed and used to correctly handle and reconnect relations and define properties. This mechanism is useful to model large and complex topologies in an abstract way and use substitutable Service Templates as a means to express subsystems. Figure 2.6 show an example where Service Template A is used as a substitution for the Node Template *NT B 3* in Service Template B. This is possible because Service Template A exposes a substitutable Node Type attribute of the same type the Node Template *NT B 3* is an instance of (Node Type X).

2.2.3 Policies

TOSCA allows to specify non-functional requirements of an application in the form of policies [OAS13]. These policies are typically meant to express Quality of Service(QoS), access control or placement aspects of *Node Templates* of a topology [OAS16]. *Node Templates* can be associated with a set of policies which specify the actual properties of these non-functional requirements. Analogous to *Node Types*, a *Policy Type* defines these properties. A *Policy Template* defines the invariant set of properties. The policy itself specifies the variant properties when a *Policy Template* is put to actual usage on a *Node Template. boundary definitions* provide a mechanism to attach *Policy Templates* to *Service Templates* to state that a certain policy is applied to the whole topology. Additionally the

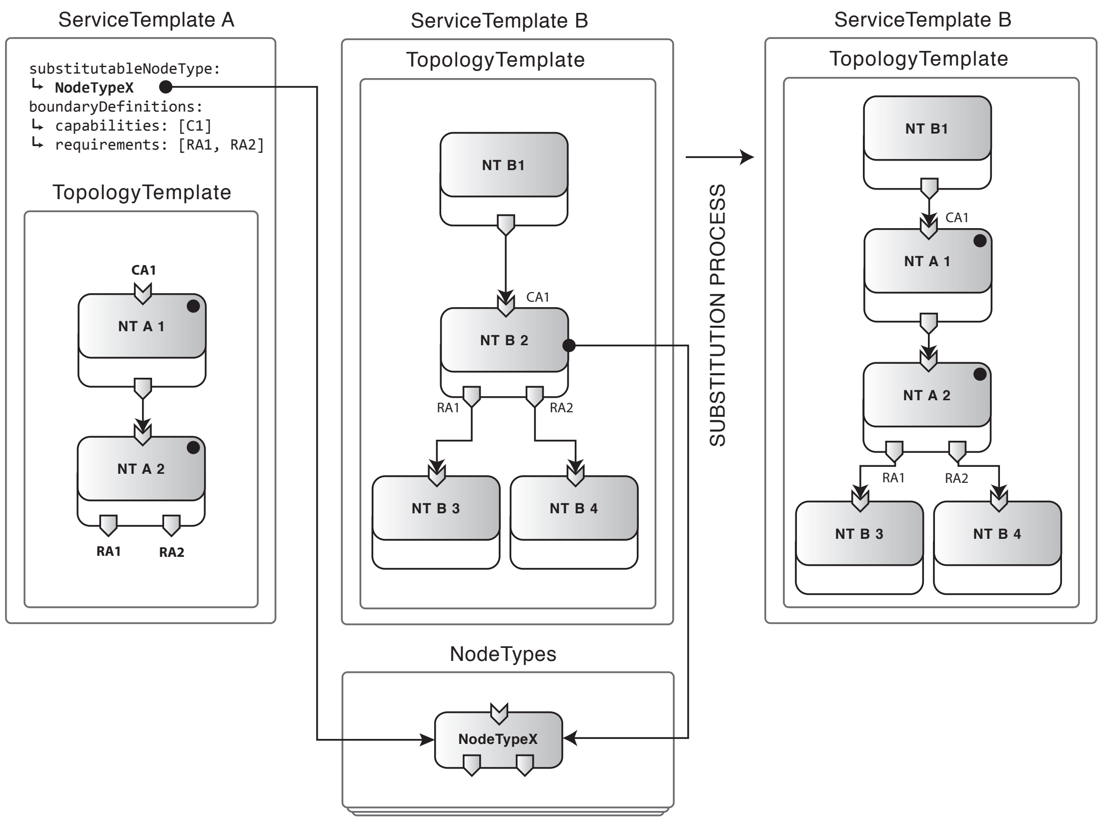


Figure 2.6: Substitution of Node Template by Service Template according to TOSCA [OAS13]

standard does not make any assumptions of a specific policy language that has to be used. Therefore policies can be utilized to express behavior that is not directly expressible via the default components of TOSCA.

2.2.4 Networking in TOSCA

The original TOSCA standard does not make any assumptions how networking should be modeled. The more recent versions of TOSCA provide dedicated networking sections [OAS16]. The network modeling approach of the TOSCA Simple Profile 1.0 is therefore used throughout this thesis. It defines that a physical or logical network can be modeled as a network Node Type. To describe that a VM is part of a network it needs to be bound to a port and this port is in turn linked to a network. Ports are modeled as Node Types too. To better understand this concept a quick look at the real world is helpful. In order to connect to a network a computer requires a Network Interface Card (NIC). This NIC is bound to a computer. To establish a connection to a network the NIC needs to be connected (linked) to a network via ethernet cables. In order to be connected to two separate networks at the same time two NICs are required. A TOSCA port Node Type can be regarded as the virtual equivalent of a NIC.

2.2.5 TOSCA Simple Profile for Network Functions Virtualization

In 2017 a simple profile for NFV was introduced that specifies a NFV specific data model using TOSCA language [OAS17]. As mentioned in Section 2.2 newer profiles are expressed using YAML but compatible with the TOSCA 1.0 XML definition. At the time of writing, the specification is still in draft state and no final version is available. TOSCA for NFV introduces a few new default Node Types that mostly align with the ETSI definitions of components in the NFV domain [OAS17] [ETS14b]. The profile is designed to be able to express any required information to specify an individual VNF or a Network Service composed of multiple VNFs in a vendor neutral fashion. This way, VNFs or complete NS, can be defined in a Service Template and packaged into a self-contained CSAR. This CSAR can then be handed to customers who in turn import it into their TOSCA compatible runtime [OAS17]. The essential building blocks, defined as Node Types are the following:

Virtual Deployable Unit (VDU) is basically a VM that specifies requirements in terms of compute resources(e.g number of CPUs and RAM) and the software image that contains the actual VNF .

Connection Point (CP) are Node Types that bind to a VDU. Each CP represents a virtual NIC of a VDU. For example a VDU of a virtual firewall needs at least two CPs (NICs) to connect to two separate networks.

Virtual Link Descriptor (VLD) represents a direct connection between two or more CPs of VDUs. This is the virtual representation of a manually built chain or sequence of physical network functions, similar to the example in Section 2.1.

This list does not fully exhaust the Node Types and Relationship Types that are introduced in TOSCA for NFV but is sufficient to express basic concepts in later chapters. A VNF is composed of at least one VDU. Network Services are in turn composed of at least two VNFs that can be linked with Virtual Link Descriptors. TOSCA for NFV can be seen as a separate TOSCA profile that directly targets NFV. NFV semantics can indeed by modeled in TOSCA without using the TOSCA profile for NFV. The remainder of this document threats VDUs as regular VM Node Types and CPs as regular ports that can be linked to networks.

2.3 OpenTOSCA Ecosystem

OpenTOSCA is a collection of open source software modules to model, deploy and manage cloud applications using TOSCA. The three main components are Eclipse Winery, the OpenTOSCA container and Vinothek [BBKL14b]. While Winery provides functionality to model TOSCA definitions in a graphical way the container provides the runtime to deploy and manage application instances. A self-service portal to enable users to instantiate and terminate instances is enabled through Vinothek.

2.3.1 Eclipse Winery

Since manual editing of complex and cross-referenced XML documents is error-prone, *Winery* provides a modern graphical interface to abstract such complexity away. For all entities defined in TOSCA, special forms and interfaces exist to edit corresponding properties and attributes. In addition to greater comfort for the user, this also allows for automatic validation to avoid incorrect input. The *Topology Modler* supports the user in creating and editing topologies based on the defined types and templates. This is achieved by rendering an editable graph of Node Templates as vertexes and Relationship Templates as edges. Node Templates automatically expose fields to enter values for their Node Type's defined properties. Additionally policies, capabilities, requirements and deployment artifacts can be assigned. Based on this graph additional validation and further processing can be done. The backend of Winery is implemented in Java and equipped with a JAX-RS based REST-API. Both previously mentions UI components (repository UI and topology modler) are modern HTML5 web applications and accessible via standard web browsers. Also both frontends are implemented as Single-Page-Applications using angular[2]. As the architecture diagram in figure 2.7 shows, all interaction between the separate frontends and the backend that processes and persists the information is solely done via the exposed REST-API. Winery uses a well-defined directory structure to store all data related to TOSCA definitions in a file-based manner instead of utilizing a separate database system. This structure is based on similar principles as the csar packing format. To encompass the storage and management of multiple *Service Templates* the use of encoded namespaces as directory names is employed. As TOSCA definitions are packaged as CSAR-files Winery provides designated import and export mechanisms.

2.3.2 Container

The OpenTOSCA Container provides the runtime necessary to create a running instance of a modeled service. To achieve this the container first needs to import a service in form of a CSAR file. The accompanying UI allows users to either import a new CSAR-file from their local filesystem or import a CSAR from a connected Winery repository. When a desired service is selected all files are extracted to be parsed, validated and stored. If the desired services does not bring imperative plans for building and managing itself, the container needs to generate plans. This is done by deriving an imperative workflow from the declarative topology definition [BBK+14]. Implementation artifacts are then deployed to the implementation artifacts engine (Tomcat Webserver). The resulting endpoints are then bound to the specific calls in the plans that reference these implementation artifacts. Finally the bound plans are deployed to the plan engine(Apache ODE). When the service is to be started, the container calls the endpoint of the build plan and tracks all resulting procedures and the instance's state. Ultimately an instance with a unique id is created and available management plans can be triggered or the instance can be terminated using the termination plan [BBH+13].

[2] https://angular.io/

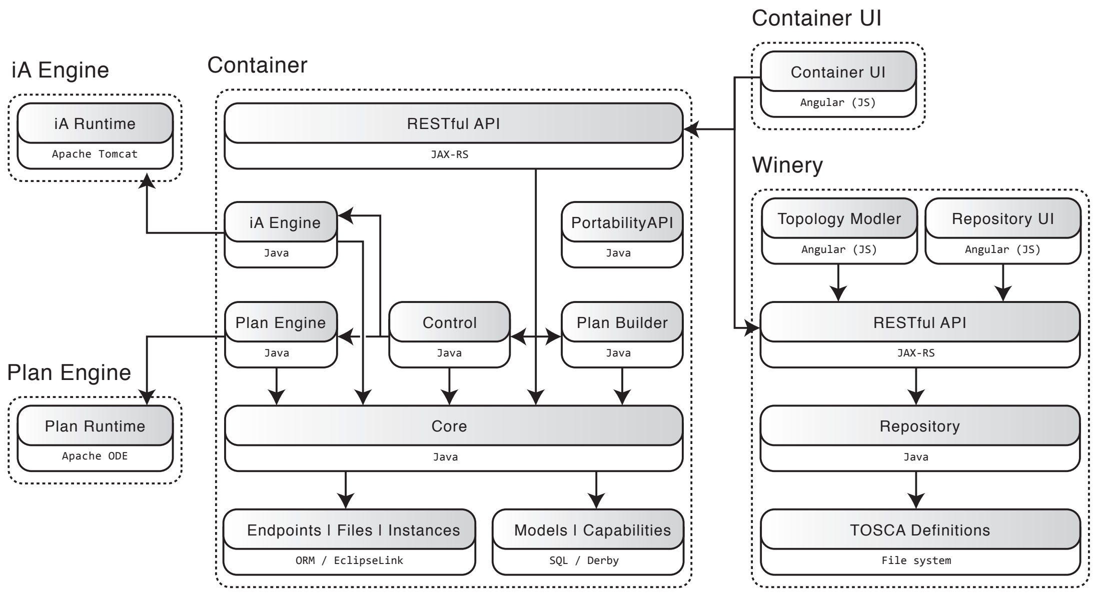


Figure 2.7: OpenTOSCA ecosystem architecture diagram

2.4 Security

The Oxford dictionary defines security as “the state of being free from danger or threat” [Oxf18]. This definition does apply to computer systems and networks alike but needs to be specified in greater detail to account for the domains.

2.4.1 Security Properties

There are many attributes that are essential to a computer system’s security. A widely used metaphor to express a system’s security is the CIA-Triangle [TC08] consisting of:

Confidentiality: Keeping things that are supposed to be private non-disclosed to other party [Sta11]

Integrity: Guarding against destruction or altering of information [Sta11]

Availability: Ensuring reliable access to ab information system [Sta11]

These three properties can be seen as the baseline for a secure system. However, the list is often extended with the following three additional properties to further define desired behavior of a system.

Non-repudiability Preventing sender or receiver from denying a message that was transmitted. This also includes to have a proof a message was sent or received [Sta11]

Authenticity: Verifying that users are who they say they are. Additionally the verification that exchanged information comes from a trusted source [Sta11]

Authorization: Granting access to specific services or resources based on permission [Sta11]

It is mandatory that a system addresses all of these properties to be considered secure. It is important to note that security is not regarded as a one time task. Each change to any given system can potentially influence one or more of these properties. Security therefore becomes a recurring task. The wide variety of different systems imposes that not all applications are created equally regarding security. Obviously military operations require a higher level of security than a simple blogging application on the internet. Nonetheless, a baseline of security is more important then ever for any kind of connected system.

2.4.2 Threats, Vulnerabilities and Risk

As the original definition of the term security in Section 2.4.1 implies the absence of threats. The term threat need to be defined accordingly. The National Institute of Standards and Technology (NIST) defines a threat as “any circumstance or event with the potential to adversely impact operations, assets, or individuals through an information system” [RKJ06]. It is important to distinguish between threat, risk and vulnerability. A vulnerability is a weaknesses in a program or software that can be exploited by threats to gain unauthorized access to an asset. Risk is the potential loss, damage or destruction of an asset as the result of a threat exploiting a vulnerability. A threat agent or threat source is an entity that has the intention of exploiting a vulnerability of a system, for example a hacker [RKJ06]. Regarding web applications there exists a wide range of typical vulnerabilities. Every year the Open Web Application Security Project (OWASP) *Top Ten Project* releases a list of the most common risks. These include injections, security misconfiguration, cross-site-scripting, the usage of components with known vulnerabilities and many more [OWA17].

Injections attacks are described as the top risk. Injection attacks are a form of code or data confusion [Sho14]. An attacker supplies a control character, followed by commands. For example, in Structured Query Language (SQL) injection, a single quote will often close a dynamic SQL statement and appends a second statement. This can potentially alter stored data, create new data or delete all data. Unix shell scripts can be targeted as well. These attacks are very common when user input is not validated or sanitized.

Cross Site Scripting vulnerabilities can be discovered in almost two third of applications [OWA17]. User input is not validated or sanitized and ends up in rendered HTML for example. This can result in the rendering of potentially malicious scripts in the HTML. These scripts are controlled by the attacker.

Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage [OWA17]. Automated scanners are useful for detecting misconfigurations, use of default accounts or configurations. These scanners are used by attackers too to check if potential misconfigurations can be automatically exploited.

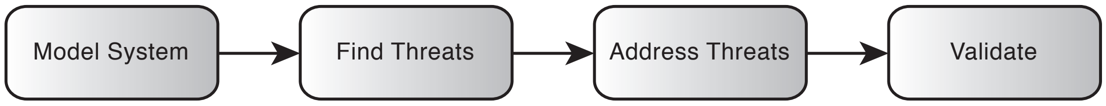


Figure 2.8: The *Four Step Framework* Threat Modeling approach according to [Sho14]

Using Components with Known Vulnerabilities is one of the most widespread risks. Component based development that uses external components like pre made software images such as databases need to be regularly updated. The same applies if the application uses legacy software that can not easily be updated or upgraded.

As NFV promises to bring advantages in terms of cost optimization and agility, the fact that network functions now run in a virtual environment imposes new threats by itself [ETS14c]. The hypervisor is a software layer between the underlying hardware platform and the virtual machines. It provides additional attack surface for hackers to gain access to VMs. Another scenario is that an infected VM can gain access to the hypervisor to compromise other VMs or randomly create new instances [BB11]. Network traffic is very hard to monitor when virtualization is used since there is no guarantee that VMs that need to communicate are placed on actual different hardware. When multiple VMs are placed on the same physical resource, communication can be handled on the hypervisor layer and therefore not involve an actual wire transfer that can be monitored easily to spot malicious anomalies [Sco17]. A *noisy neighbor* is a common problem on shared virtualized resources. In this scenario a malicious VM consumes all available resources to perform a Denial of Service (DoS) attack on a target VM [Sco17]. All these threats are directly related to the core idea of virtualization and therefore don't target NFV specifically. The remainder of this thesis recognizes the existence of this problem but focuses on the investigation of the security benefits that can be achieved by utilizing NFV.

2.4.3 Threat Modeling

Threat Modeling enables the assessment of threats in a structured way. The OWASP describes Threat Modeling as the task to identify, communicate, and understand threats and mitigations within the context of protecting something of value [OWA18]. According to Shostask, Threat Modeling is conducted using the *Four Step Framework* (depicted in Figure 2.8) that maps to four key questions that need to be asked [Sho14]:

What are we building? Model the system that needs to be built, deployed or changed.

What can go wrong? Find threats using that model.

What are we going to do about that? Address these threats.

Did we do a good enough job? Validate the approach.

Threat category	Violated property
Spoofing	Authenticity
Tampering	Integrity
Repudiation	Non-repudiability
Information disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

Table 2.1: STRIDE: threats and desired security properties

STRIDE is a mnemonic and framework for threat modeling. The acronym that stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege. Each threat type violates a corresponding security property (see Section 2.4.1). A mapping between the threat types of STRIDE and the violated security properties is displayed in Table 2.1. The STRIDE approach to threat modeling was invented by Loren Kohnfelder and Praerit Garg [KG99]. It was designed to help people developing software identify the types of attacks that software tends to experience [Sho14]. Variants of STRIDE are STRIDE-per-element and STRIDE-per-interaction. STRIDE-per-element focuses on the threats of each element in the model while STRIDE-per-interaction focuses on the interaction between elements.

A combination of both is desirable. Using the STRIDE approach a security expert can enumerate the things that might go wrong on each element of the model. The result of this assessment is then captured and assigned to the element or interaction. Alternatives to STRIDE include attack trees [Sch99] or the usage of attack libraries. The attack tree approach represents the attacks against a system in a tree structure where the root node represents the goal and all leafs represent different ways to achieve that goal [Sch99]. Attack libraries such as Common Attack Pattern Enumeration and Classification (CAPEC) consist of a large set of detailed threat descriptions. CAPEC currently holds 577 individual attack patterns [Bar08]. Due to the fact that STRIDE is more abstract than attack libraries it provides more freedom in regards of threat modeling and can be used to express any kind of threat, present in a library or not.

2.4.4 Network Security Requirements

The importance of deriving network security requirements from the application that needs to be secured is best explained by looking at a modified version of the running example. The original running example depicts a typical cloud application where different components of the application can be scaled individually. The modified version, illustrated in Figure 2.9 of the topology consists of the same software components but all are hosted on a single VM instead of being distributed over two VMs. Both versions need to publicly expose the HTTP port (typically port 80) of the web server to the internet in order to allow users to connect to the site. Although the same software components are used it has vastly different implications for the network. Considering the modified version, basic network security can be achieved by simply blocking all inbound traffic that targets any port except 80. This is

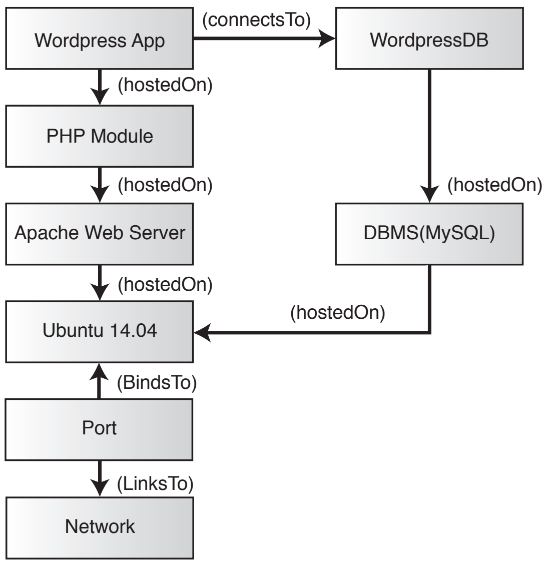


Figure 2.9: Modified version of the running example where all software components are hosted on a single VM

due to the fact that communications between the WordPress app and the database happen inside the same VM. The original version instead relies on two VMs communicating over a network. If both VMs would publicly expose all ports that are needed for communicating this would result in a publicly exposed database attackers can target. In order to achieve basic security for the original version, traffic between VMs has to be allowed while traffic entering the network has to be limited. This is a trivial example where it is assumed that both VMs exist on the same network. The whole process becomes more complex if the more services and networks are involved. It is a common practice to start with multiple services on a single VM and later decompose an application in order to scale services independently. This also underlines the point made in Section 2.4 that keeping a system secure is a recurring task.

3 Related Work

Achieving secure NFV utilization but also enabling security by means of TOSCA are approaches that are targeted in current research. The following chapter acknowledges the work that was done prior to this thesis.

3.1 Management and Orchestration

A variety of Management and Orchestration(MANO) systems already exist that align with the ETSI NFV specification and NFV reference architecture (see Section 2.1.2). Namely OpenBaton[1], ETSI's own Open Source Mano(OSM)[2], ONAP[3], T-NOVA[4] or the Tacker plugin[5] for Openstack. Tacker represents the only project that supports TOSCA for VNF on-boarding and export. TOSCA is used just as a means of exchanging VNFs in a vendor independent way and not for actual instantiation. Tacker uses an translator that translates TOSCA definitions to HEAT[6], Openstack's internal orchestration format. Since Tacker is a part of the Openstack ecosystem, NFV orchestration benefits from the integrated tenant-isolation that Openstack provides natively. In therms of security, all of these projects focus on securing the deployment and management of NFs and VNFs. Basic measures are employed like user accounts and passwords. T-NOVA provides a module called *Gatekeeper* that ensures secure access to interfaces of all kind of VNFs[**t-nova**].

The SecMANO

3.2 Security-aware Modeling and Deployment

The authors of [WWB+13] introduce *Policy4TOSCA*, a framework that enables security-aware modeling and deployment of cloud applications based on TOSCA policies. A formal policy definition based on a taxonomy defining the stage, layer, and effect of policies is introduced. Multiple policies are combined into an offering together with a formal TOSCA Cloud service definition. An offering represents a specific level of security of an application. The authors provide an example consisting of a demo application that has the following offerings: *full security*, *encrypted database* and *default(no security enhancement)*.

[1] https://openbaton.github.io/
[2] https://osm.etsi.org/
[3] https://www.onap.org/
[4] http://www.t-nova.eu/
[5] https://wiki.openstack.org/wiki/Tacker
[6] https://wiki.openstack.org/wiki/Heat-Translator

Each offering achieves the level of enhanced security by providing either a set of modified plans (build, management and termination) which enforces the necessary additional steps or by deploying modified implementation artifacts that implement the security features. According to the individual security requirements of the use case a customer can then choose a fitting offering.

Kepes et al. expand on *Policy4TOSCA*[WWB+13] by leveraging policies of the TOSCA standard to generate policy-aware imperative build plans [KBF+17]. Components need to be annotated with policies, for example a *secure password policy* that should be used on a database component. The model containing components and corresponding policies is then processed by a plan generator. Policies are used to determine which additional steps need to be taken while the desired provisioning of an application. The *secure password policy* example needs to verify that a password for a database is secure enough. Since this password is or can be set during runtime, the generated plan can check that just in time when the component is about to be provisioned. This way different levels of security can be achieved by annotating security-related policies on components that are enforced during provisioning. This approach lays the groundwork for investigating automated security enhancement of application topologies in the realm of TOSCA.

3.3 NFV as a Security Enabler

Farris et all. present a framework for integrating security features enabled by NFV and SDN in an IoT scenario [FBT+17]. The authors investigate the utilization of SDN/NFV-based security function that can manage malicious traffic and in turn enable zoned networks. This is achieved by a custom designed orchestration layer that is used to specify network policies that need to be enforced by different security technologies. The research was conducted in the scope of the ANASTACIA project[7] that aims to bring security to IoT and cyber-physical systems.

The authors of [RK18] introduce the concept of so called complex services. These services are composed of cloud application components and NFV components. They imply that such combined services can benefit the actors that use these services in forms of cost reduction and security. It is highlighted that topologies that include components from separate families (NFV and cloud applications) are currently hard to provision. This is due to the fact that both types of components impose different requirements. For example, traditional cloud applications don't have packet processing as a main concern while it's the main subjective of NFV. Besides that, application components need to be orchestrated in a specific order to ensure connectivity and NFV topologies have the need for the chaining of services in a predefined order. The authors propose a candidate solution to accommodate the problems by combining the functionality of the SONATA project[8] with of Terraform[9]. Here SONATA is used as network service development and orchestration

[7]http://anastacia-h2020.eu
[8]http://www.sonata-nfv.eu/
[9]https://www.terraform.io/

platform while Terraform supplies a structured multi-cloud orchestration solution that can handle cross-cloud dependencies. Finally the authors conclude that the combination of NFV and application topologies is a viable but currently widely overlooked approach.

4 Concept

First an overview of the concept is presented. A list of requirements that need to be satisfied is compiled in Section 4.2. The concept leverages threat modeling as a base for security assessment of application topologies and describes how the required information can be expressed using TOSCA in Section 4.3. NFV based threat mitigation is detailed in Section 4.4. Section 4.5 explains the benefits that can be achieved by utilizing abstraction. Subsequently, limitations regarding the proposed concept are discussed. A summary of the concept concludes the chapter.

4.1 Overview

An overview of the proposed concept is described using the following scenario, involving all relevant entities, domains, processes. Additionally the assumptions that were made are detailed. We assume that an application architect (further referred to as architect) is tasked to design an application. The application should be complemented with VNFs to enhance the security. We further assume a repository with concrete VNF implementations and configurations exists. This repository will be referred to as solution space and represents a library of potentially deployable VNFs. It is assumed the architect does not have the required domain knowledge, which VNF should or can be utilized to mitigate a security threat. However the architect is able to identify security threats present in the application and it's components. We assume there is a security expert that has the required knowledge to classify a concrete VNF implementation as an appropriate countermeasure to one or more given threats. To be able to recommend VNFs to the architect on how to mitigate the present threats in the application a convenient mechanism is required. The proposed concept aims to establish a relation between the potential threats in an application and the concrete VNFs in the solution space. In order to do so the security expert is required to create a catalog of threats. Subsequently VNFs in the solution space are marked as potential countermeasures to one or more threats of this catalog. This way the architect is enabled to annotate the components of the application with corresponding threats from the catalog. The annotated threat can then be compared to the solution space in order to recommend an appropriate VNF. When new potential threats are discovered by the architect or security expert the catalog needs to be extended. This in turn requires the security expert to assess the solution space for VNFs that can mitigate the newly discovered threat and mark them accordingly. In the case no existing VNF can mitigate a threat, additional VNFs need to be imported to the solution space or the threat must be accepted or resolved using a different approach. Abstraction of concrete VNF implementations allows for the usage of VNFs in application topologies without taking the implementation details into account. The security expert therefore groups similar concrete VNFs that share the same semantics or just differ in configuration in an abstract VNF. For example a virtual firewall, with different configurations available, represents multiple entries in the solution space. All variants can be expressed as the same abstract VNF. The resulting abstract VNF only exposes the information relevant to the architect when modeling. Modeling application topologies with abstract VNF components additionally allows for easy substitution of concrete implementations without changing the original topology. The combination of information thus created gains all involved parties to gain the following insights: It is assessable which threats are present in the application if components are annotated with threats from the catalog. Based on these threats recommendations can be made which abstract VNF should be used in the topology to mitigate these threats. When abstract VNFs are used in the topology it can be derived if a corresponding concrete VNF renders the present threats mitigated.

4.2 Requirements

The following list of requirements is compiled to guide the creation of a valid proposal. Later the concept is evaluated based on the same requirements.

Requirement 1 (R1): The concept must be conform with the TOSCA standard and hence only use already available entities and mechanisms

Requirement 2 (R2): The concept must respect the proposed TOSCA for NFV draft

Requirement 3 (R3): The concept must use NFV in a way that enhanced security can be achieved

Requirement 4 (R4): Security requirements need to be directly derived from the application that shall be protected

Requirement 5 (R5): The concept shall employ a structured approach to solve the problem employing industry standard practices

4.3 Finding Threats in Application Topologies

This concept leverages threat modeling as a means to assess threats of applications in a structured manner. Threat modeling represents an industry standard practice and is widely used by security professionals to determine the current security state of an application. The Four Step Framework (see Section 2.4.3) is applied and appropriate means are investigated to express all required information using TOSCA.

4.3.1 Modeling the System

TOSCA and threat modeling are both based on the principle to model the components of a system. TOSCA uses a model to describe the structure of an application in order to deploy and manage the application. Threat modeling uses a model to assess the application components and their interactions for possible threats. Threat modeling software tools like OWASP threat dragon[1] or Microsoft's Threat Modeling Tool[2] provide graphical user interfaces to assist the user to create a model of an application. The user can add processes or components, interaction between these components, so-called trust boundaries and threat agents to the diagram. This diagram can then be assessed for threats. Using this approach a user that currently uses TOSCA for modeling would need to create a separate diagram of all components in such a software tool. A TOSCA Service Template already consists of a topology template that describes all components of an application and the relationships between them. A topology template can be used to graphically render a topology graph [KBBL13]. This allows to use the actual components that make up the deployment of an application as the model for threat modeling. Using the TOSCA model as the source for threat modeling eliminates the possibility of having a possibly mismatching separate model. This has the benefit of no required additional effort to create redundant diagrams.

4.3.2 Finding and Describing Threats

When threat elicitation should be performed on a model using the STRIDE approach the findings need to be assigned to the model's components and interactions. We define the term *threat descriptor* as a means to capture the required information of a threat present in the model. Each threat needs to have a unique name to be distinguishable from other threats. The type of a threat needs to be defined according to STRIDE. A description is required to provide detailed information how a threat imposes risk on a system. Additionally, as it's common practice, the severity of a threat needs to be expressible. A component or interaction of the model is the target of a threat, so this needs to be captured as well. Summarizing these findings a threat descriptor can be defined as the following set:

$$threatdescriptor \coloneqq \{Name, STRIDE - Type, Description, Severity, Target\}$$

The threat descriptor needs to be expressed using TOSCA language to be used in a TOSCA based model. TOSCA does not have the concept of a threat, but TOSCA does have policies that can be used to express non-functional aspects (Section 2.2.3). A threat represents a non-functional aspect. A Policy Type is a reusable entity with properties definition to account for required information. Policy Templates are instances of a Policy Type and are used to annotate Node Templates in a topology. To be able to express a threat descriptor using TOSCA we first introduce a new Policy Type with the name *Security.Threat*. We use a subset of the threat descriptor to define the Policy Type's properties. Only the STRIDE type, description and severity need to be defined here. In TOSCA properties of an entity can be defined as custom XML elements. Listing 4.1 shows how the properties for the *Security.Threat* Policy Type are defined. Enumeration is used to restrict the possible values for STRIDE and severity. For description string based input is allowed.

The XML-based definition of the introduced *Security.Threat* Policy Type that uses the custom XML element as properties definition is shown in Listing 4.2.

[1] https://threatdragon.org
[2] https://docs.microsoft.com/de-de/azure/security/azure-security-threat-modeling-tool

Listing 4.1 XML-based definition for the properties definition of the Security.Threat Policy Type

```
<xs:simpleType name="strideEnum" final="restriction" >
    <xs:restriction base="xs:string">
        <xs:enumeration value="Spoofing" />
        <xs:enumeration value="Tampering" />
        <xs:enumeration value="Repudiation" />
        <xs:enumeration value="Information disclosure" />
        <xs:enumeration value="Denial of Service" />
        <xs:enumeration value="Elevation of Privilege" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="severityEnum" final="restriction" >
    <xs:restriction base="xs:string">
        <xs:enumeration value="low" />
        <xs:enumeration value="middle" />
        <xs:enumeration value="high" />
    </xs:restriction>
</xs:simpleType>
<xs:element name="ThreatProperties">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="STRIDE" type="strideEnum"/>
            <xs:element name="Severity" type="severityEnum"/>
            <xs:element name="Description" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Listing 4.2 XML-based TOSCA definition for the Security.Threat Policy Type

```
<PolicyType
  name="Security.Threat"
  abstract="no"
  final="yes"
  targetNamespace="http://opentosca.org/security/threat">
  <PropertiesDefinition element="ThreatProperties" />
</PolicyType>
```

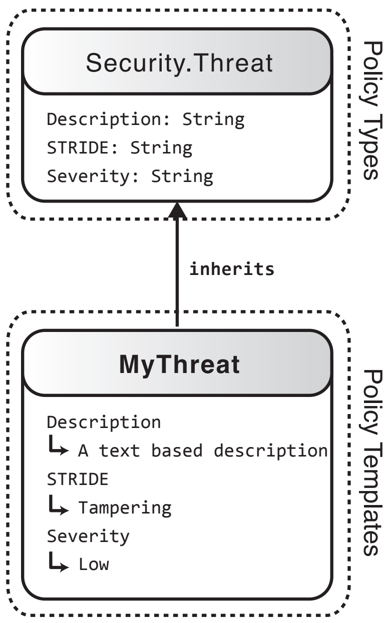


Figure 4.1: Security.Threat Policy Type and an exemplary Policy Template instance

The threat descriptor mandates a unique name for each threat. TOSCA uses a XML qualified name (QName) to uniquely define and reference a TOSCA definition. When a Policy Template is created this QName needs to be specified. This fact is leveraged to account for unique names of threats. This enables the creation of a Policy Template that is an instances of the *Security.Threat* Policy Type. This Policy Template contains the information of a unique name, the STRIDE type, a severity rating and a textual description. A graphical representation of an example Policy Template that defines this is illustrated in Figure 4.1

The last missing piece of information to fully qualify as threat descriptor is the target. The Policy Template intentionally does not contain any information about the target of a specific threat. This way the Policy Template becomes a reusable entity in itself. Policy Templates are meant to be attached to Node Templates. To indicate that a Node Template is targeted by a threat, a Policy Template of the predefined *Security.Threat* Policy Type is attached to a Node Template. This way all required information of the threat descriptor is supplied by the resulting model.

For each threat present in an application it is now possible to create a corresponding Policy Template that is attached to the targeted Node Template. To illustrate the result of this process the running example is annotated with potential threats (see Figure 4.2). Threat Policy Templates are visualized as warning signs. Only the names of the threats are shown, the rest is omitted for brevity.

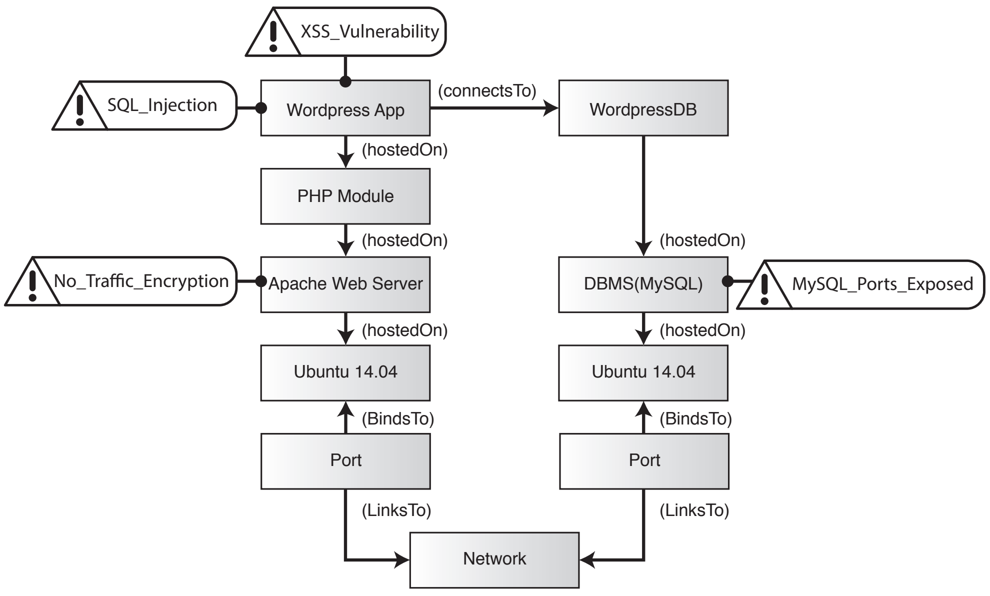


Figure 4.2: Running example topology assessed using the STRIDE method with annotated threats

Algorithm 4.1 Finding all threats in a Service Template

```
1: function GETTHREATS(serviceTemplate)
2:     $threats \leftarrow \emptyset$
3:     for all $policies$ in NodeTemplates of serviceTemplate do
4:         for all $policy$ in $polcies$ do
5:             $policyType \leftarrow policy.type$
6:             if $policyType = Security.Threat$ then
7:                 $policy.target \leftarrow NodeTemplate$
8:                 $threats \leftarrow$ ADD$(policy.template, threats)$
9:             end if
10:        end for
11:    end for
12:    return $threats$
13: end function
```

As a result of this phase, all threats present in a application topology can be retrieved as valid threat descriptors. A simple iteration over all Node Templates and determining if a Node Template is annotated with a Policy Template of the Policy Type *Security.Threat* is sufficient to do so. Algorithm 4.1 describes this procedure in a pseudo code notation.

4.4 Mitigating Threats with NFV

NFV uses a component based approach to network functions. Each network function is expressed as a virtual component. We assume that a specific network function can have the ability to mitigate one or more specific threats. For example we found that the running example has the threat of unencrypted web traffic targeting the web application. A VNF that encrypts web traffic is therefore a potential mitigation for this threat. This raises the need to describe the relation between a specific threat and VNF components that potentially mitigate the threat. The TOSCA for NFV draft defines that each deployable VNF is stored in a separate Service Template. This represents the need to establish a relation between a Threat described as Policy Template and a specific Service Template containing a VNF.

We can not attach the same threat Policy Template to the Service Template to indicate that a Service Template is an appropriate mitigate for this threat. This is due to the intended use of threat Policy Templates to represent present threats.

To describe the relation between a threat and a concrete VNF we introduce a new *Security.Mitigation* Policy Type. There is no dedicated mechanism to define relationships between Policy Templates like it's the case with Node Templates. To circumvent this a *ThreatReference* property that is represented by a QName is defined. The Policy Type definitions is described in Listing 4.3 and the corresponding properties definition XML element is shown in Listing 4.4.

Listing 4.3 XML-based TOSCA definition for the Security.Mitigation Policy Type

```
<PolicyType name="Security.Mitigation" abstract="no" final="yes" targetNamespace="http://
opentosca.org/security/mitigation">
  <PropertiesDefinition element="MitigationProperties" />
</PolicyType>
```

Listing 4.4 XML-based definition for the properties definition of the Security.Mitigation Policy Type

```
<xs:element name="MitigationProperties">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ThreatReference" type="xs:QName"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

For each threat Policy Template there needs to be a mitigation Policy Template that references the threat. Referencing is achieved by providing the QName of the threat Policy Template for the *ThreatReference* property on the mitigation Policy Template. A naming convention is established regarding mitigations. The name of a mitigation Policy Template

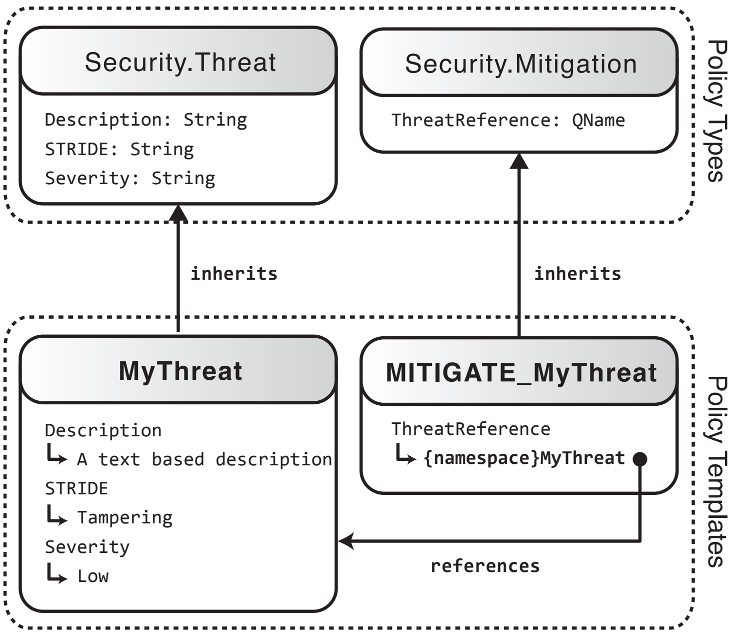


Figure 4.3: Security.Mitigation Policy Type and a corresponding Policy Template that references a threat

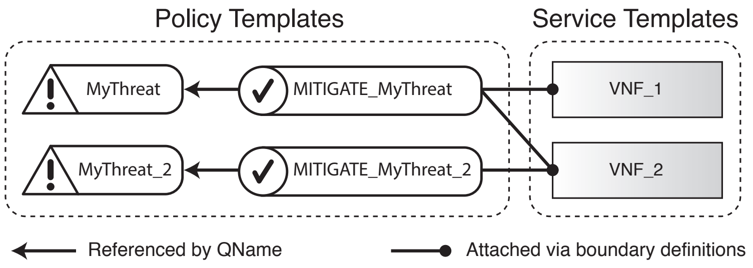


Figure 4.4: VNF Service Templates with attached mitigation Policy Templates and referenced threat Policy Template

should be the same as the threat Policy Template prepended with "MITIGATION_". An example of all this is illustrated in Figure 4.3. The naming convention allows for easier handling of pairs of Policy Templates without looking at the individual defined properties.

To put the created mitigation Policy Template to use it needs to be attached to a VNF. Since a VNF is stored in a Service Template this is achieved by defining according boundary definitions. These boundary definitions allow to attach Policy Templates to Service Templates exactly like it's the case for Node Templates inside a topology. Figure 4.4 shows an example of two VNF Service Templates. Here "VNF_1" is able to mitigate "MyThreat" while "VNF_2" can mitigate "MyThreat" and "MyThreat_2".

4.5 Abstracting Network Function Details

The application architect should be able to use a VNF without knowing about its implementation details. TOSCA allows to substitute an abstract Node Template with the topology of a Service Template. In order for this to work a Service Template needs to expose the information for which Node Type it can be used as a substitute (see Section 2.2.2).

A naive approach would be to create an abstract Node Type for each VNF implementation. This is highly inefficient and defeats the purpose of abstraction because each Service Template is tightly coupled to exactly one corresponding abstract Node Type.

This concept mandates that abstract Node Types should be used as a grouping mechanism for similar implementations. For example an abstract *Firewall* Node Type can be used as an umbrella to define properties, requirements and capabilities that account for typical implementations. Gateway firewalls for example need to be correctly connected to an external network and an internal network to properly function. This is required to be expressed by two separate requirements in the boundary definitions of a Service Template that map to the correct internal ports.

An abstract Node Type with the name of *S-VNF* is introduced. *S-VNF* stands for security relevant VNF. *S-VNF* will be used as a root entity which all security relevant VNFs derive from. This has the drawback that a single abstract Node Type can not accurately express all possible requirements any VNF might expose. To account for a greater degree of freedom this concept introduces the convention that abstract Node Types need to be created for capturing grouping semantics and inherit from S-VNF. This provides a middle ground between S-VNF and actual VNF implementation. Users are free to model these Node Types however they want as long they are abstract and inherit from S-VNF.

As a result, inheritance of S-VNF indicates that the all VNF implementations, that define a substitutable Node Type of a S-VNF group are indeed security relevant functions. Figure 4.5 depicts the resulting structure of the relation between S-VNF, S-VNF groups and VNF implementations.

4.6 Recommending S-VNF Groups for Threat Mitigation

Based on the threat Policy Templates present in an application topology and properly described VNFs (attached mitigation Policy Templates and S-VNF group Node Type as substitutable Node Type) recommendations can be made.

First a list of threats present in an application is compiled. How this can be achieved is detailed in Section 4.3.2. Subsequently a list of all VNFs is gathered. A Service Template qualifies as a VNF if it exposes a substitutable Node Type that is an ancestor of S-VNF. Both lists are then analyzed for intersections of matching pairs of threats and mitigations. This results in a list of abstract S-VNF group Node Types that can be used in the application topology to mitigate present threats. Algorithm 4.2 shows a pseudo code implementation of this approach and returns all present threats with corresponding possible mitigation Node Types. Finally the abstract S-VNF Node Template in the topology can be substituted with VNF implementations to get a deployable model.

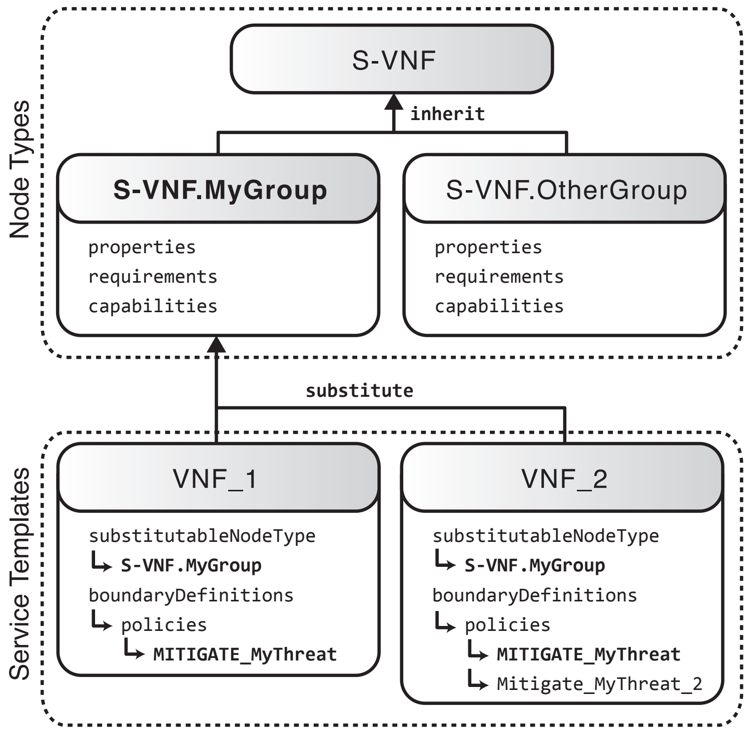


Figure 4.5: VNF Service Templates as substitutable Node Types for an abstract S-VNF group

If VNF implementations exist that are able to mitigate a present threat we can say that a threat can be mitigated by including a corresponding S-VNF group in the topology. If this specific S-VNF is somehow present in the topology we can a assume a threat is successfully mitigated when the abstract S-VNF group is substituted before deployment.

4.7 Limitations

The proposed concept has some limitations due to the structure of TOSCA, the nature of NFV and the wide range of possible threats. Policies do provide a possibility to describe non-functional semantics, such as threats, in the context of TOSCA, but only Node Types and Service Templates (via boundary definitions) can be targeted. Relationships between components, such as *connects to* can therefore not be annotated. Since relations can only exist between components, one way to circumvent this is to annotate the target or source of the relation to express the presence of a threat.

Threat modeling software such as the options discussed in Section 4.3.1 provide a way to graphically represent trust boundaries and threat agents in the model to assist the user to understand the system. This concept uses the topology template of a TOSCA Service Template as a model. A topology that is modeled using TOSCA is intended to be deployed.

Algorithm 4.2 Compiling a VNF candidate list for threat mitigation

```
1: function findAbstractMitigations(serviceTemplate)
2:     $candidateVNFs \leftarrow$ GetAllVNFServiceTemplates()
3:     $threats \leftarrow$ getThreats($serviceTemplate$)
4:     for all $threat$ in $threats$ do
5:         $threat.mitigations \leftarrow \emptyset$
6:         for all $VNF$ in $candidateVNFs$ do
7:             if $VNF.policy$ references $threat$ then
8:                 $threat.mitigations \leftarrow$ Add($threat.mitigations, VNF.substitutableNodeType$)
9:             end if
10:        end for
11:    end for
12:    return $threats$
13: end function
```

Threat agents represent non-deployable entities. It is possible to create Node Types and Relationship Types that don't do anything in regards to orchestration and use these to represent entities and their relations to the components just for modeling purposes. This approach would defeat the general purpose of TOSCA and results in topologies where it has to be determined what is deployable and what isn't.

To determine if a threat is mitigated it is currently checked if an instance with a matching mitigation Policy Template is present in the topology. The sheer presence of a mitigation does not guarantee the right placement in the topology. For example a proxy firewall needs to be placed between two components. If this firewall is placed at the perimeter of the network instead, the functionally is different than intended.

VNFs can be utilized as components that secure other components as stated in Section 2.4.1. This is only applicable to threats that involve the networking layer. If an application component stores sensitive information in an insecure way, no VNF is capable of changing this fact. This is just an example to show that a lot of threats can't directly be mitigated by simply using a NFV based approach to enhance security.

4.8 Summary

The proposed concept utilizes the STRIDE approach to threat modeling. Therefore a *Security.Threat* Policy Type is introduced to create reusable threat Policy Templates. These threat Policy Templates are instances of *Security.Threat* and are attachable to Node Templates of an application topology. This indicates which Node Templates are targets of which threats. To establish a connection between threats and VNF implementations that could potentially mitigate these threats, a *Security.Mitigation* Policy Type is introduced. Each mitigation Policy Template references a single corresponding threat Policy template. These mitigation Policy Templates are then attached to the Service Templates of appropriate VNF implementations via boundary definitions.

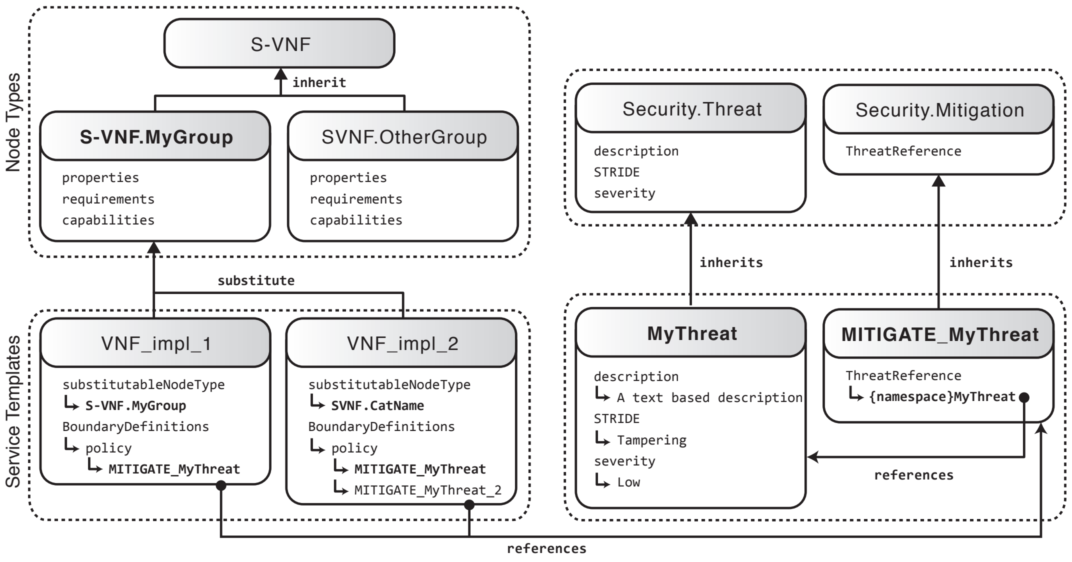


Figure 4.6: Summary of all introduced Node Types, Policy Templates and their relations

A *S-VNF* Node Type is introduced to act as an umbrella of all possible abstract VNF groups. A VNF group needs to be an abstract Node Type that captures the properties, requirements and capabilities of VNF implementations that share the same external attributes. A S-VNF group is required to inherit from S-VNF. A VNF Service Template needs to expose matching properties, requirements and capabilities via boundary definitions and declare that is can be used as a substitute for a certain VNF group by defining a substitutable Node Type.

The resulting construct of all introduced Node Types, Policy Types, exemplary Policy Templates and their intended relations, is illustrated in Figure 4.6.

5 Implementation and Validation

This chapter first details the prototypical implementation of the concept. A validation of concept and reference implementation are done.

5.1 Eclipse Winery Extensions

Since the proposed concept concerns the modeling aspects, the proof of concept is implemented in the context of Eclipse Winery (see Section 2.3.1). The Java based backend of the Winery, described in Section 2.3, was extended to allow for the consumption of user input to generate valid pairs of threats and mitigations. Additionally the logic to find Service Templates that satisfy the mitigation requirements for threats of an application topology was implemented. The following work was done in a newly created, independent module and all data that is exposed or consumed is done so by following RESTful semantics to comply with current practices. Winery provides two frontend components for users to interact with the backend in a graphical way. To allow for interactions with the newly created backend extensions the frontend components were modified accordingly.

5.1.1 Backend: Creating Threat Mitigation Pairs

The possibility to manually create Policy Types and Policy Templates is already present in the current Winery version to comply with basic TOSCA functionality. To be able to threat model and find appropriate VNFs we defined the need to have pairs of threats and mitigations, where the mitigation references the threat. Since the creation of a mitigation Policy Template can be error-prone because the correct threat has to be manually referenced by it's QName, it was opted to create a custom REST-API-Endpoint that handles the synchronous creation of both, the threat Policy Template and the corresponding Mitigation Policy Template. The endpoint expects values for the defined property definition of a threat and the desired name. This allows for input validation to enforce correct and required properties. The given name is sanitized to have a valid QName later. A Policy Template of type Security.Threat is created, the desired properties assigned and stored. The generated QName is used to create a Policy Template of type Security.Mitigation and the local part of the QName is prepended with "MITIGATION_" to satify the proposed naming convention. The resulting REST-API-Endpoint is accessible at "/winery/threats/" of a winery instance. A POST request to the resource with the required parameters creates a threat and corresponding mitigation.

5.1.2 Backend: Threat Catalog

A threat catalog ensures that threats that were modeled in the scope of other applications become accessible and reusable. Since all threat Policy Templates are instances of the predefined Policy Type *Security.Policy* all available threats can be easily discovered by iterating over all Policy Templates in a given repository and check if they are instances of the said type. The threat catalog is accessible at the same endpoint mentioned in Section 5.1.1 but with a HTTP GET request to comply with REST semantics. Finally a list of all threats is rendered as JavaScript Object Notation (JSON) to allow for easy consumption in the frontend of the application.

5.1.3 Backend: Threat Assessment and Mitigation Recommendation

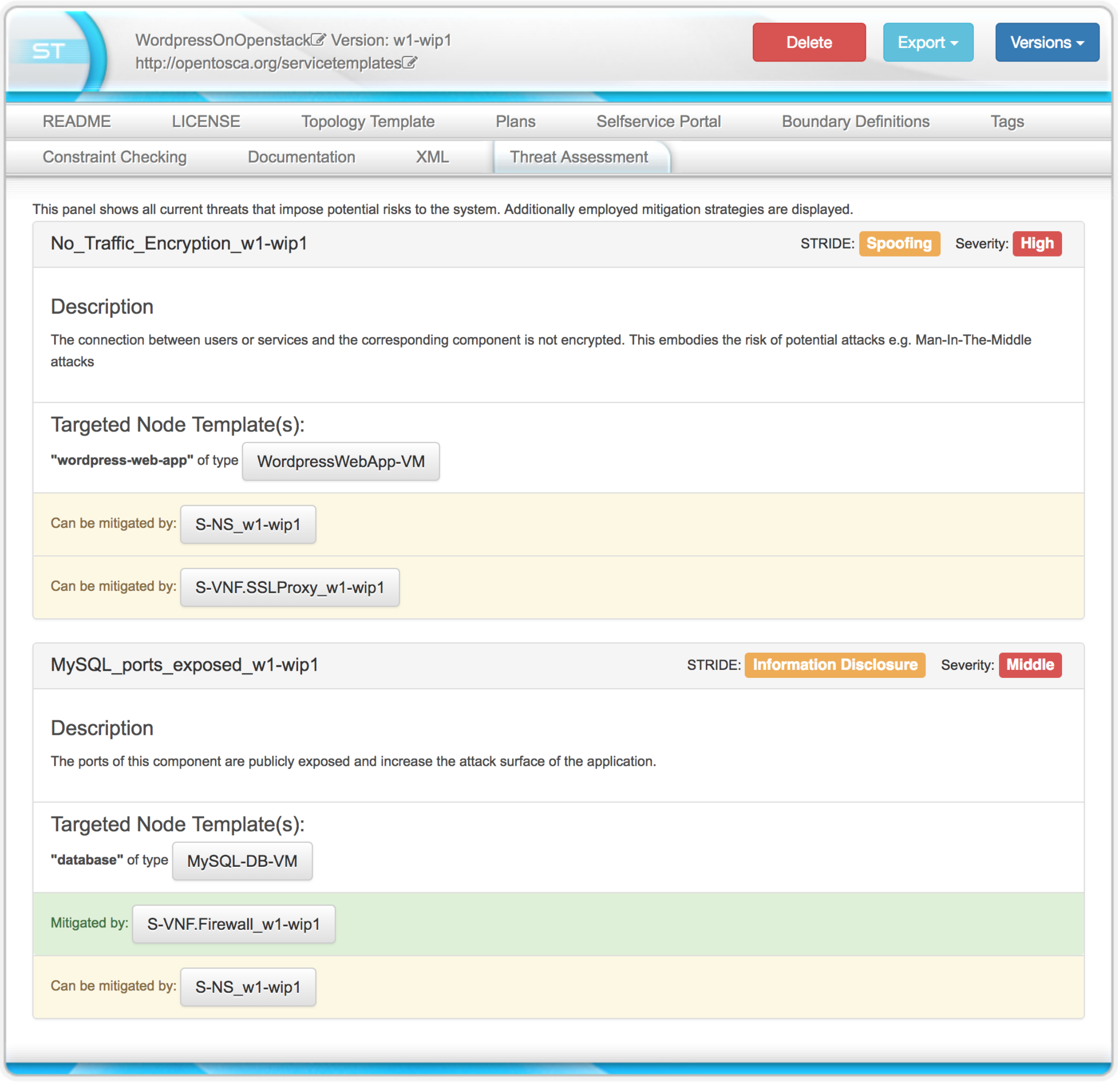
Threats are annotated to Node Templates in the form of Policy Templates. To assess the security state of a Service Template there needs to be an endpoint that lists all threats present in a Service Template . This is achieved by iterating over all available Node Templates in a Service Template and compiling a list of Policy Templates that match the criteria. Afterwards all Service Templates in the repository are filtered for matching mitigation Policy Templates defined in the boundary definitions and it is checked for proper inheritance. This results in a list of concrete implementations of VNFs. To make a recommendation what abstract category of VNF can be used to mitigate threats the substitutable Node Type is referenced. The API endpoint that returns all computed results is put under the Service Template REST resource since threat assessment is done regarding a specific Service Template at “winery/servicetemplates/<namespace>/<ID>/threatassessment”.

5.1.4 Backend: Modifications of the Substitution Module

Winery already has a basic substitution feature. This feature enables the substitution of Node Templates in a topology with Service Templates. The process is automated and a find first strategy is utilized. So when an abstract Node Type is present in a topology it is substituted with the first Service Template that presents a fitting substitutable Node Type. The substitution process aims to connect all previous relations to the abstract Node Type to the inserted Service Template topology. The current implementation does not address edge cases such as multiple Relationship Templates of the same Relationship Type. Multiple Relationship Templates of the same type are essential to NFV topologies since a single Node Template can be connected to multiple other Node Templates with the same type of relation (see Section 4.5). The implementation of the substitution module was therefore modified to account for relationships that target or source requirements of Node Templates.

5.1.5 Repository UI: Threat Assessment

Threat assessment is incorporated in the detail view of a given Service Template in the repository UI. The existing navigation menu was extended with a threat assessment view. The threat assessment view requests all available threats in a Service Template by querying



This panel shows all current threats that impose potential risks to the system. Additionally employed mitigation strategies are displayed.

No_Traffic_Encryption_w1-wip1 STRIDE: Spoofing Severity: High

Description

The connection between users or services and the corresponding component is not encrypted. This embodies the risk of potential attacks e.g. Man-In-The-Middle attacks

Targeted Node Template(s):

"wordpress-web-app" of type WordpressWebApp-VM

Can be mitigated by: S-NS_w1-wip1

Can be mitigated by: S-VNF.SSLProxy_w1-wip1

MySQL_ports_exposed_w1-wip1 STRIDE: Information Disclosure Severity: Middle

Description

The ports of this component are publicly exposed and increase the attack surface of the application.

Targeted Node Template(s):

"database" of type MySQL-DB-VM

Mitigated by: S-VNF.Firewall_w1-wip1

Can be mitigated by: S-NS_w1-wip1

Figure 5.1: Threat assessment of a Service Template in the Winery repository UI

the appropriate REST API endpoint described in Section 5.1.3. In addition to the display of threats their corresponding mitigation options (if available) are suggested. If a threat can be mitigated (an appropriate VNF implementation exists) the UI indicates this by displaying a message and coloring the background yellow. If a Node Template of a suggested mitigation is present in the topology the threat is considered mitigated and indicated by a respective message and green background. If recommendation can be made (meaning no VNF is available that can mitigate the threat) the UI displays a message and colors the background of the threat red. An example of the created UI is depicted in Figure 5.1.

5.1.6 Topology Modler: Threat Creation, Mitigations and Threat Catalog

A modal was added to the topology modeler to allow for easy creation of threat mitigation pairs when modeling the application topology. The UI provides a form with input fields that directly map to the properties definitions of the *Security.Threat Policy Type*. Submitting the form results in a HTTP POST request to the API endpoint specified in Section 5.1.1. Additionally all available threats (threat catalog) is rendered below to allow for efficient reuse. The threat assessment view described in Section 5.1.5 is displayed in the modal too. This is done to enable the user to directly add the recommended mitigations to the topology with a designated button. The complete modal is pictured in Figure 5.2.

5.2 Validation

The presented concept is first validated based on the requirements specified in Section 4.2 and afterwards validated based on the processing of an example service that mimics the running example.

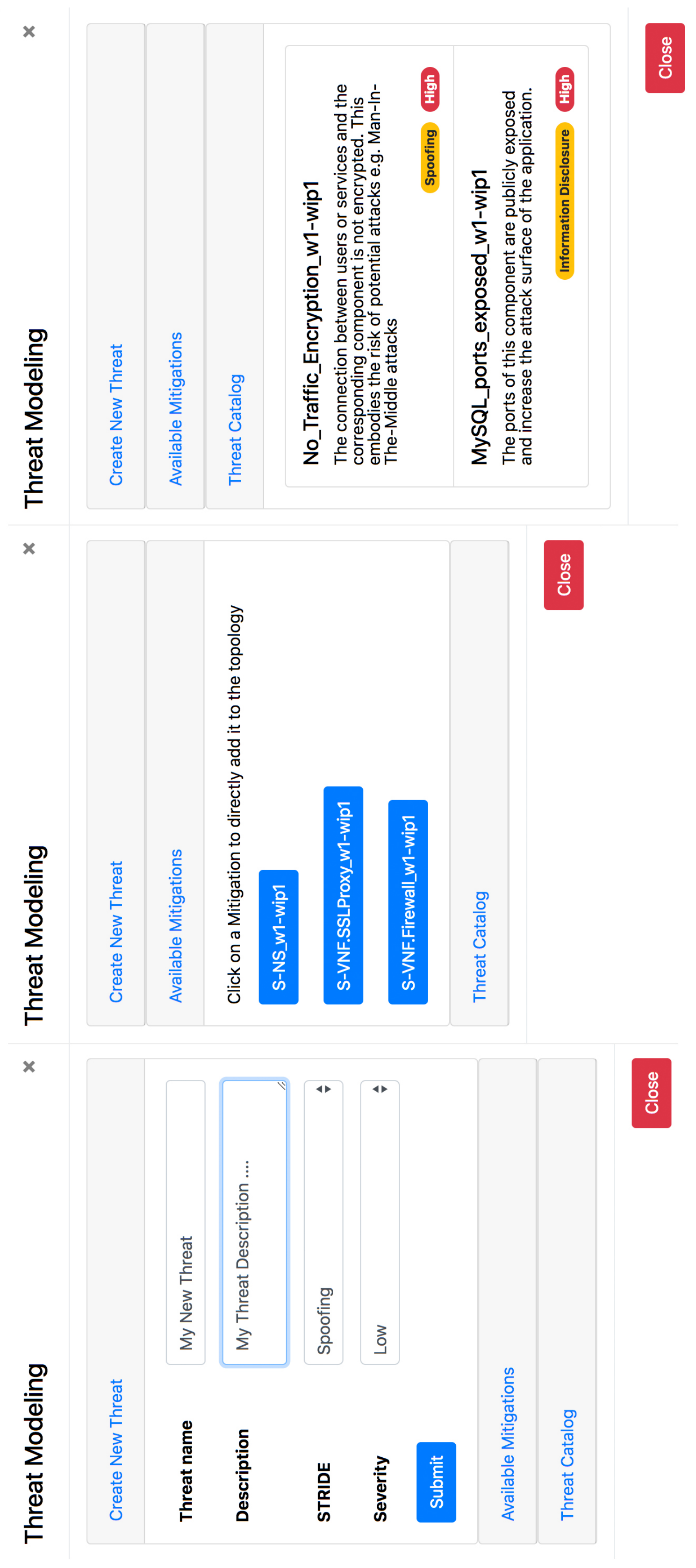
5.2.1 Requirements

In Section 4.2 a list of requirements was compiled in order to guide the creation of the concept and validate it afterwards. In the following, the individual requirements are discussed. The results of the validation are summarized in Table 5.1. To rate the satisfaction of each requirement plus signs are used. Two plus signs represent full satisfaction while a single plus sign indicates that the requirement is mainly satisfied. For each single plus sign a detailed explanation is provided. The requirements are referenced by their numbers and compressed descriptions.

The requirement of TOSCA conformity is regarded as fully satisfied. The concept is based on the introduction of special Node Types and Policy Types. These are standard elements of the TOSCA language and defined accordingly. No additional entities were introduced

The requirement of respecting the TOSCA profile for NFV is seen as satisfying when some assumptions are made. The TOSCA profile for NFV is not finalized yet. As long as standard Service Templates are used to express deployable VNF implementations the proposed concept is considered valid. Even if the profile might introduce additional default Node Types that are used in a VNF model, these need to be contained in the Service Template of the VNF. This is due to the fact that this concept does not make assumptions how a VNF is modeled as long it is contained in a Service Template. This implies the correct exposure of its requirements, properties and capabilities via boundary definitions. Users are then tasked to provide appropriate abstract Node Types and ensure it inherits from S-VNF.

The requirement of using NFV to enhance security is regarded as fully satisfied. The proposed concept establishes a connection between threats and VNFs in order to mitigate these threats. NFV is therefore utilized as a main driver for security enhancement.



Threat Modeling ×

Create New Threat

Threat name	My New Threat
Description	My Threat Description ....
STRIDE	Spoofing
Severity	Low

Submit

Available Mitigations

Threat Catalog

Close

Threat Modeling ×

Create New Threat

Available Mitigations

Click on a Mitigation to directly add it to the topology

S-NS_w1-wip1

S-VNF.SSLProxy_w1-wip1

S-VNF.Firewall_w1-wip1

Threat Catalog

Close

Threat Modeling ×

Create New Threat

Available Mitigations

Threat Catalog

No_Traffic_Encryption_w1-wip1

The connection between users or services and the corresponding component is not encrypted. This embodies the risk of potential attacks e.g. Man-In-The-Middle attacks

Spoofing High

MySQL_ports_exposed_w1-wip1

The ports of this component are publicly exposed and increase the attack surface of the application.

Information Disclosure High

Close

Figure 5.2: Threat modeling options in the Winery topology modler

Requirement	Satisfaction
R1("TOSCA conformity")	++
R2("TOSCA For NFV Profile")	+
R3("NFV as security enabler")	++
R4("Security requirements derived from application")	++
R5("structured approach")	++

Table 5.1: Concept validation based on the initially defined requirements. Two plus signs indicate full satisfaction. Single plus signs represent satisfaction with additional remarks

Threat name	No_Traffic_Encryption	MySQL_ports_exposed
STRIDE type	Spoofing	Information Disclosure
Severity	High	Middle

Table 5.2: Detailed overview of the properties of the two modeled threat Policy Templates

Threat modeling ensures that threats are assessed in a structured way and is considered industry standard practice. These threats are then directly annotated to application topology components. This accounts for satisfaction of the requirement to derive security requirements directly from the application topology. Additionally the requirement regarding the structured approach is satisfied by applying the STRIDE method to threat modeling.

5.2.2 Example Service

Figure 5.3 visualizes the example service that is composed of a simplified version of the running example and modeled in Eclipse Winery.

A demo repository of VNFs is created to simulate the presence of multiple different VNF implementations. Figure 5.4 illustrates an exemplary firewall VNF implementation that is composed of a single VM Node Type and two Port Node Types on the right. This Service Template will be referenced as *Firewall_VFN*. A VNF implementation that acts as a Secure Sockets Layer (SSL) proxy VNF is shown on the left and will be referenced to as *SSLProxy_VFN*. Each VNF is contained in its own Service Template and composed of one VM and a varying number of ports. Each Service Template exposes boundary definitions that address the requirements and capability of internal Node Templates. For example the port of the SSLProxy_VNF needs to be linked to a network and therefore exposes a requirement. This requirement is exposed in the boundary definitions. A *S-VNF.Firewall* Node Type and a *S-VNF.SSLProxy* Node Type were created to act as S-VNF groups. Both of those inherit from S-VNF. The S-VNF group Node Types define the requirements and capability accordingly. For the demo a subset of threats was modeled using the threat modeling UI. The resulting threat Policy Templates are described in Table 5.2. The description property is omitted for brevity. Both mitigation Policy Templates were created automatically.

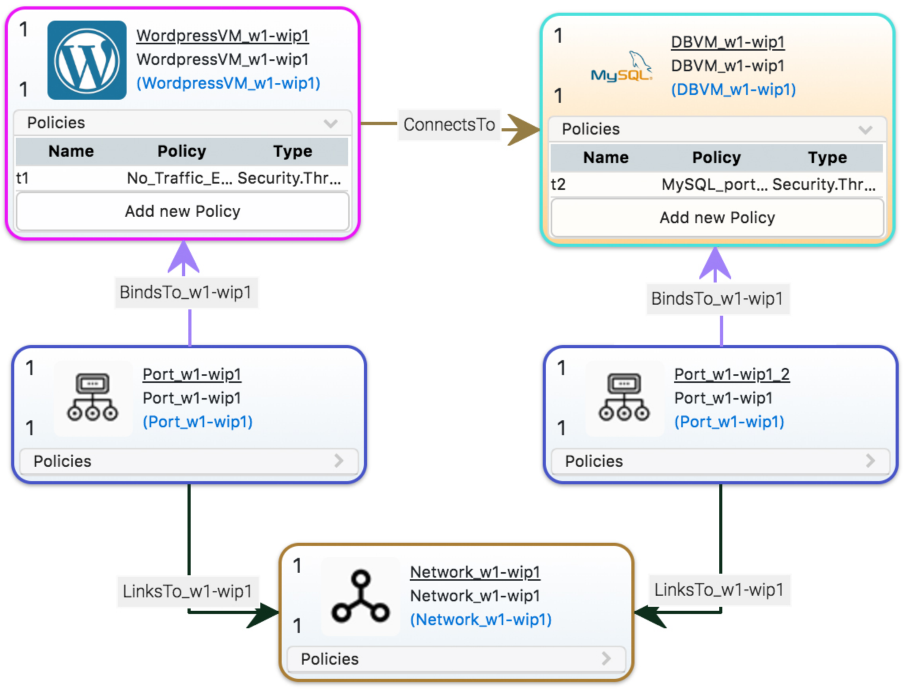


Figure 5.3: Simplified running example topology modeled in Eclipse Winery with attached threat Policy Templates

The *No_Traffig_Encryption* threat is attached to the WordpressVM Node Template of the demo topology while the *MySQL_ports_exposed* is attached to the DBVM Node Template. The generated corresponding mitigation Policy Templates are attached to the VNF implementations. The SSLProxy_VFN is able to encrypt traffic and therefore the *MITIGATE_No_Traffig_Encryption* is attached via boundary definitions. The *MITIGATE_MySQL_ports_exposed* is attached to the boundary definitions of the Firewall_VNF Service Template. Subsequently the implemented system correctly recognizes the threats present in the demo topology and suggests the usage of the according S-VNF groups. The threat assessment UI correctly recommends that in order to mitigate the *No_Traffig_Encryption* the S-VNF.SSLProxy group should be included in the topology. Additionally it is recommended that the S-VNF.Firewall Node Type should be used to mitigate the threat *MySQL_ports_exposed*. The suggested S-VNF group Node Types (S-VNF.Firewall and S-VFN.SSLProxy) are inserted into the topology and connected as the requirements demand. This is illustrated in **??**. A look at the threat assessment UI then correctly states that both threats can be considered mitigated. Finally the substitution procedure is triggered and the resulting topology contains the correctly connected topology of application components and concrete VNF components. The final topology is illustrated in Figure 5.6

During the validation the possibility of false positives was discovered when the system evaluates which threat are effectively mitigated. For example if two threats (threat1 and threat2) are present in a topology. Two VNFs are assumed to be available and VNF1

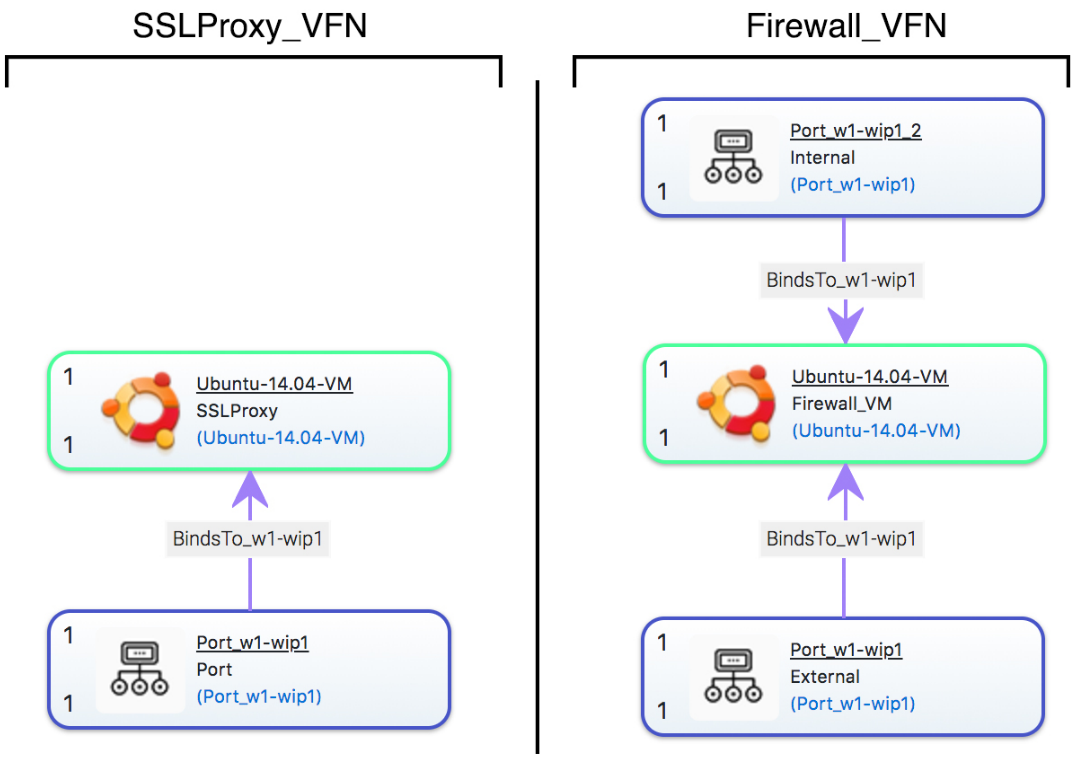


Figure 5.4: Topology of a demo VNF implementations that represents a SSL proxy (left) and topology of a demo VNF implementations that represents a firewall (right)

mitigates threat1 and VNF2 mitigaes threat2. Both VNFs belong to the same S-VNF group (through the declaration of substitutable Node Types). Based on the proposed concept this means that both threat can be mitigated, which is true. When the corresponding S-VNF group Node Type is placed in the topology it is assumed that both threats are mitigates because the Node Template can potentially substituted with both VNFs. This assumption is false since only one VNF will end up in the topology through substitution. This results in the exclusive mitigation of either threat1 or threat2.

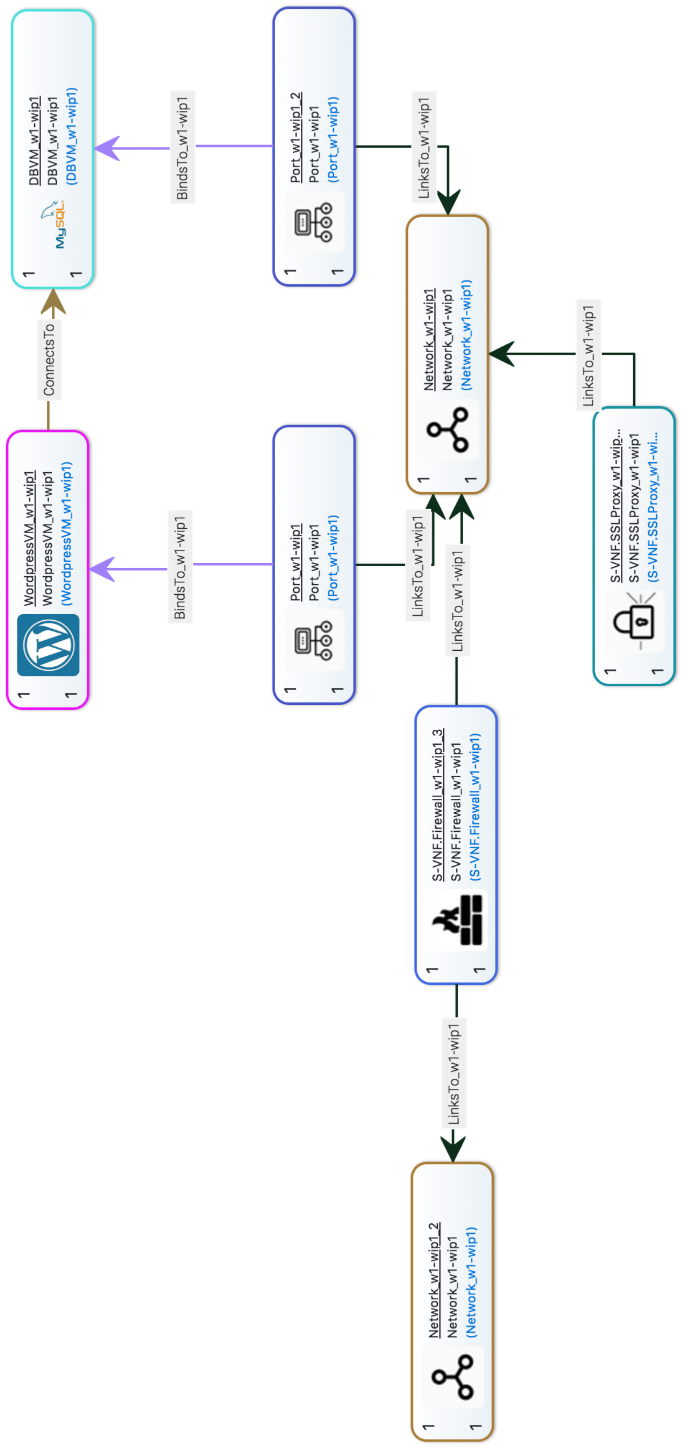


Figure 5.5: Demo topology enhanced with recommended SVFN groups (S-VNF.Firewall and S-VFN.SSLProxy)

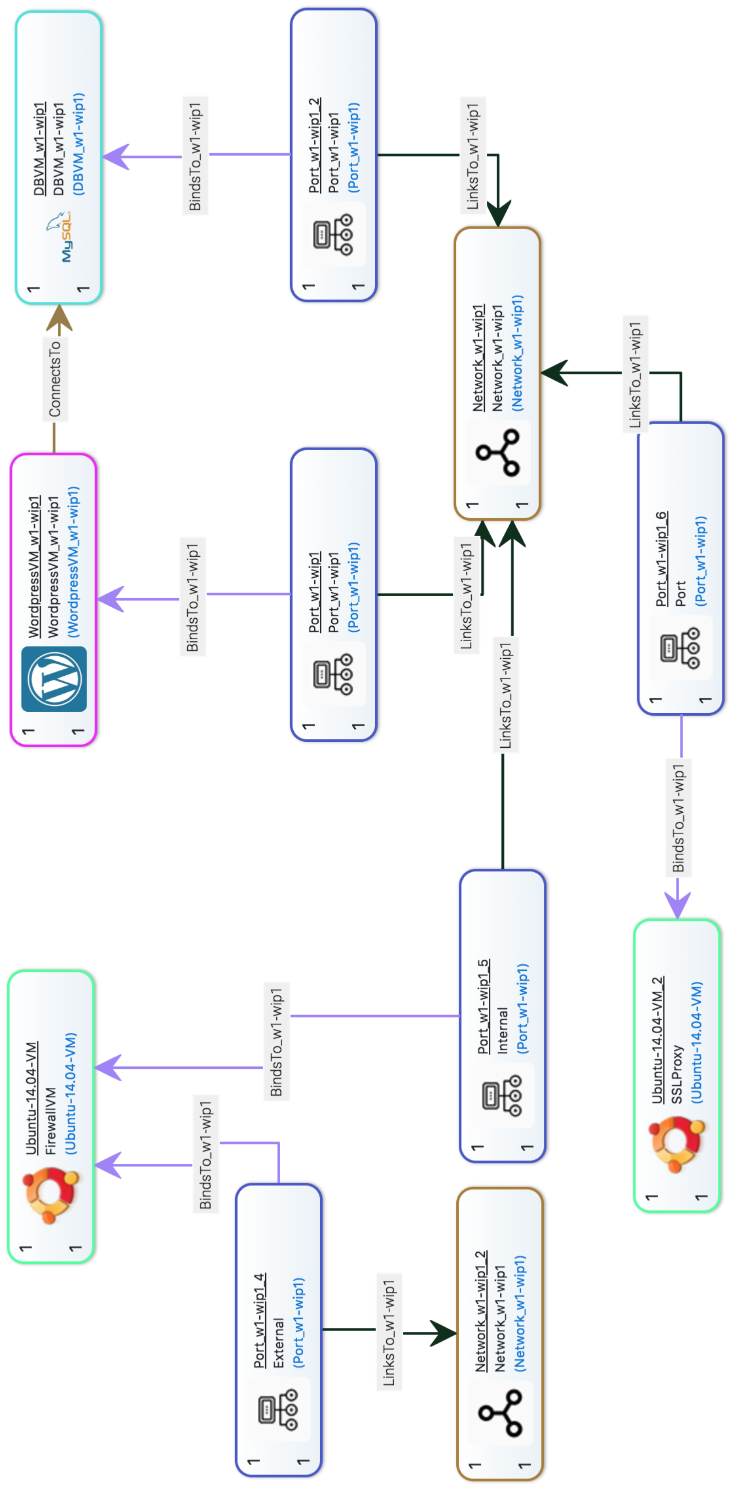


Figure 5.6: Resulting final topology after substitution including application components and concrete VNF implementations

6 Conclusion and Future Work

The goal of this thesis is to develop a concept for security-aware modeling and deployment of NFV topologies using TOSCA. The proposed concept employs threat modeling as an industry standard practice to assess security threats of application topologies. Virtual network functions are levered as countermeasures to these threats. The developed approach establishes a relation between possible threats and appropriate NFV based countermeasures. This is done by leveraging standard TOSCA policy mechanisms. Dedicated threat and mitigation Policy Types are introduced and defined to express the required information. Additionally an abstract *S-VNF* Node Type was introduced to function as a root ancestor for all security related VNF categories and concrete implementations. The developed approach mandates that for each concrete VNF implementation there needs to exist an appropriate abstract representation. This in turn enables an abstract NFV modeling approach. The application architect does not have to care for the specificities of a particular VNF implementation. The TOSCA concept of substitution is brought in to refine abstract models with concrete VNF implementations that match the countermeasure requirements for present threats. Based on the information carried by the introduced Policy Types and Node Types, automated recommendations can be made how an application topology can be protected with and NFV based approach. This way the knowledge gap between application architects and network security experts can be bridged in an automated manner.

To validate the approach, a proof-of-concept was implemented as an extension to the TOSCA modeling tool Eclipse Winery. The prototype allows for the provisioning of all necessary Policy Types and Node Types to use the proposed concept. The creation of threats automates the creation of corresponding threat and mitigation Policy Templates. A graphical interface enables users to assess the current security state of an application (what threats currently exist and which can be or are mitigated). The topology modeling tool was modified to enable the user to directly add appropriate abstract VNFs to the topology with the click of a button. An extension to the substitution algorithm incorporates NFV specifics in a standard conforming way.

Future work regarding the concept should include further research on how the correct placement of VNFs can be guaranteed. The current concept only allows to check if a required VNF is present in a topology and ignores the fact where it is. For example if a VNF that encrypts traffic is deployed in a cloud environment but the target is not in the same network the encryption is breachable. This is vital for providing secure end-to-end encryption. Additional sources for potential future work can be found in the limitations section of the concept (Section 4.7). The possibility of false positive estimates regarding the effective mitigation of threats discovered in the validation (see Section 5.2) needs to be addressed as well.

Regarding the implementation, future work needs to be done in terms of visualization and creation of Relationship Templates. The already mentioned extension to the substitution module accounts for a correct implementation of TOSCA relations. The current functionality needs to be extended to allow for relations that target or source a concrete requirement of a Node Template.

Bibliography

[Bar08] S. Barnum. “Common attack pattern enumeration and classification (capec) schema description”. In: *Cigital Inc, http://capec. mitre. org/documents/documentation/CAPEC_Schema_Descr iption_v1* 3 (2008) (cit. on p. 33).

[BB11] M. A. Bamiah, S. N. Brohi. “Seven deadly threats and vulnerabilities in cloud computing”. In: *International Journal of Advanced engineering sciences and technologies* 9.1 (2011), pp. 87–90 (cit. on p. 32).

[BBH+13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, S. Wagner. “OpenTOSCA -- A Runtime for TOSCA-Based Cloud Applications”. In: *Service-Oriented Computing*. Ed. by S. Basu, C. Pautasso, L. Zhang, X. Fu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 692–695. ISBN: 978-3-642-45005-1 (cit. on p. 29).

[BBK+14] U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann, J. Wettinger. “Combining declarative and imperative cloud application provisioning based on TOSCA”. In: *Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014* (2014), pp. 87–96. DOI: 10.1109/IC2E.2014.56 (cit. on p. 29).

[BBKL14a] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann. “TOSCA: portable automated deployment and management of cloud applications”. In: *Advanced Web Services*. Springer, 2014, pp. 527–549 (cit. on pp. 23, 26).

[BBKL14b] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann. “Vinothek-A Self-Service Portal for TOSCA.” In: *ZEUS*. Citeseer. 2014, pp. 69–72 (cit. on p. 28).

[Ber17] H. Berghel. “Equifax and the Latest Round of Identity Theft Roulette”. In: *Computer* 50.12 (Dec. 2017), pp. 72–76. DOI: 10.1109/mc.2017.4451227. URL: https://doi.org/10.1109/mc.2017.4451227 (cit. on p. 17).

[CCW+12] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng, D. Telekom, U. Michel. *Network Functions Virtualisation,An Introduction, Benefits, Enablers, Challenges & Call for Action*. 2012. URL: http://portal.etsi.org/NFV/NFV%7B%5C_%7DWhite%7B%5C_%7DPaper.pdf (cit. on pp. 17, 20).

[Ehr17] J. M. Ehrenfeld. “WannaCry, Cybersecurity and Health Information Technology: A Time to Act”. In: *Journal of Medical Systems* 41.7 (May 2017), p. 104. ISSN: 1573-689X. DOI: 10.1007/s10916-017-0752-1. URL: https://doi.org/10.1007/s10916-017-0752-1 (cit. on p. 17).

[ETS14a] ETSI. “Network Functions Virtualisation (NFV); Architectural Framework”. In: (2014). URL: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf (cit. on pp. 21, 22).

[ETS14b] ETSI. "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV". In: (2014). URL: https://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_nfv003v010201p.pdf (cit. on pp. 20, 28).

[ETS14c] ETSI. *NFV Security/Problem Statement. Report ETSI GS NFV-SEC 001 (V1. 1.1), October 2014*. 2014. URL: https://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/001/01.01.01_60/gs_NFV-SEC001v010101p.pdf (cit. on p. 32).

[ETS17] ETSI. "Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; VNF Package specification". In: (2017). URL: https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/004/02.03.01_60/gs_nfv-sol004v020301p.pdf (cit. on p. 21).

[FBT+17] I. Farris, J. B. Bernabe, N. Toumi, D. Garcia-Carrillo, T. Taleb, A. Skarmeta, B. Sahlin. "Towards provisioning of SDN/NFV-based security enablers for integrated protection of IoT systems". In: *2017 IEEE Conference on Standards for Communications and Networking, CSCN 2017* (2017), pp. 169–174. DOI: 10.1109/CSCN.2017.8088617 (cit. on p. 36).

[Gou15] R. K. Goutam. "Importance of Cyber Security". In: *International Journal of Computer Applications* 111.7 (2015) (cit. on p. 17).

[IHS17] IHS Markit. "The Internet of Things : a movement , not a market". In: (2017), p. 9. URL: https://cdn.ihs.com/www/pdf/IoT%7B%5C_%7Debook.pdf (cit. on p. 17).

[KBBL13] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann. "Winery--a modeling tool for TOSCA-based cloud applications". In: *International Conference on Service-Oriented Computing*. Springer. 2013, pp. 700–704 (cit. on p. 41).

[KBF+17] K. Képes, U. Breitenbücher, M. P. Fischer, F. Leymann, M. Zimmermann, U. Breitenb, M. P. Fischer, F. Leymann, M. Zimmermann. "Policy-Aware Provisioning Plan Generation for TOSCA- based Applications ". In: (2017) (cit. on p. 36).

[KG99] L. Kohnfelder, P. Garg. "The threats to our products". In: *Microsoft Interface, Microsoft Corporation* (1999) (cit. on p. 33).

[MSG+15] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. De Turck, R. Boutaba. "Network Function Virtualization: State-of-the-art and Research Challenges". In: *IEEE Communications Surveys and Tutorials* 18.1 (Sept. 2015), pp. 236–262. DOI: 10.1109/COMST.2015.2477041 (cit. on pp. 17, 19, 20, 22, 23).

[OAS13] OASIS. "Topology and Orchestration Specification for Cloud Applications". In: March (2013), pp. 1–114. URL: http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.html (cit. on pp. 18, 23, 25–27).

[OAS16] OASIS. *TOSCA Simple Profile in YAML Version 1.0. Edited by Derek Palma, Matt Rutkowski, and Thomas Spatzier*. 2016. URL: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.html (cit. on pp. 23, 26, 27).

[OAS17] OASIS. *TOSCA Simple Profile for Network Functions Virtualization (NFV) Version 1.0 Committee Specification Draft 04*. 2017. URL: http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html (cit. on pp. 18, 28).

[OWA17] OWASP. *Application Security Risks-2017. Open Web Application Security Project (OWASP).* 2017. URL: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (cit. on p. 31).

[OWA18] OWASP. "Application Threat Modeling". In: (2018). URL: https://www.owasp.org/index.php/Application_Threat_Modeling (cit. on p. 32).

[Oxf18] Oxford. *Security Defition.* 2018. URL: https://en.oxforddictionaries.com/definition/security (cit. on p. 30).

[RK18] H. Razzaghi Kouchaksaraei, H. Karl. "Joint Orchestration of Cloud-Based Microservices and Virtual Network Functions". In: (2018). URL: https://arxiv.org/pdf/1801.09984.pdf (cit. on p. 36).

[RKJ06] R. S. Ross, S. W. Katzke, L. A. Johnson. *Minimum security requirements for federal information and information systems.* Tech. rep. 2006 (cit. on p. 31).

[Sch99] B. Schneier. "Attack trees". In: *Dr. Dobb's journal* 24.12 (1999), pp. 21–29 (cit. on p. 33).

[Sco17] S. Scott-hayward. *Guide to Security in SDN and NFV.* 2017. ISBN: 978-3-319-64652-7. DOI: 10.1007/978-3-319-64653-4. URL: http://link.springer.com/10.1007/978-3-319-64653-4 (cit. on pp. 22, 32).

[Sho14] A. Shostack. *Threat modeling: Designing for security.* John Wiley & Sons, 2014 (cit. on pp. 31–33).

[Sta11] W. Stallings. *Network security essentials : applications and standards.* 2011, xiii, 366 p. ISBN: 0130160938 (cit. on pp. 30, 31).

[TC08] A. K. Talukder, M. Chaitanya. *Architecting secure software systems.* Auerbach publications, 2008 (cit. on p. 30).

[w3t18] w3techs. *Wordpress statistics.* 2018. URL: https://w3techs.com/technologies/details/cm-wordpress/all/all (cit. on p. 23).

[WWB+13] T. Waizenegger, M. Wieland, T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, B. Mitschang, A. Nowak, S. Wagner. "Policy4TOSCA : A Policy-Aware Cloud Service Provisioning Approach to Enable Secure Cloud Computing ". In: *Meersman R. et al. (eds) On the Move to Meaningful Internet Systems: OTM 2013 Conferences. OTM 2013. Lecture Notes in Computer Science, vol 8185. Springer, Berlin, Heidelberg* (2013) (cit. on pp. 35, 36).

All links were last followed on October 5, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature