Analyzing and Validating Virtual Network Requests

Jorge López[1], Natalia Kushik[1], Nina Yevtushenko[2] and Djamal Zeghlache[1]
[1]*SAMOVAR, CNRS, Télécom SudParis, Université Paris-Saclay, 9 rue Charles Fourier 91011 Évry, France*
[2]*Department of Information Technologies, Tomsk State University, Lenin str. 36, Tomsk, Russia*

Keywords: Network Virtualization Platforms, Validation, User Request Analysis, Scalable Representations.

Abstract: In this paper, we address platforms developed to provide and configure virtual networks according to user's request and needs. User requests are, however, not always accurate and can contain a number of inconsistencies. The requests need to be thoroughly analyzed and verified before being applied to such platforms. We consequently identify some important properties for the verification and classify them into three groups: a) functional or logic issues, b) resource allocation/dependency issues, and c) security issues. For each group, we propose an effective way to check the request consistency. The issues of the first group are checked with the use of scalable Boolean matrix operations. The properties of the second group can be verified through the use of an appropriate system of logic implications. When checking the issues of the third group, the corresponding string analysis can be utilized. All the techniques discussed in the paper are followed by a number of illustrating examples.

1 INTRODUCTION

Virtual networks are growing in usage and popularity along with the progress being achieved in the area of communication technologies. At the same time, virtual networks represent one of the main concepts of the future generation networks, such as 5G (ETSI, 2013). Therefore, the software and hardware components of such systems need to be thoroughly tested and verified.

Virtualization mechanisms open up the possibilities to provide any types of networks 'on-demand'. Network services and their service chains can thus be provided to users according to their preferences. Users request (virtual) networks with specific nodes and service function chains from service providers that host the desired virtual network. One of the platforms for providing such network services was developed in our previous works (Mechtri et al., 2016). This platform is a service function chain orchestrator capable of provisioning, hosting resources, deploying and instantiating the user's requested service chains (and virtual network).

As the requests for providing network services are written by users (or tenants), they can contain inconsistencies as well as semantic errors. In the platform selected for this study, the requests are expressed in the Topology and Orchestration Specification for Cloud Applications (TOSCA) language (Palma et al., 2016). In a well formatted TOSCA request (correct syntax, i.e., parsed successfully), a number of semantic inconsistencies can be present. If an inconsistent user request (a virtual network) that for example, contains contradicting declarations, is deployed in the platform, unexpected or undesirable results might occur. Different constraints might be violated, such as for example service level agreements. As the consistency of the system is compromised by such requests, in one way or another the request is not properly implemented or it can even threaten the security and safety of the whole system. This is the reason why tenant requests should be carefully verified prior to their implementation.

We note that some of the requests can contain specific dependencies between the parameters of the virtual machines being requested and the tasks (services) assigned for these machines together with the order of their execution (invocation).

In this paper, we discuss how the consistency of a user request can be checked before the platform executes such a request. In this paper, we identify three possible groups issues, which can occur in the requests, i.e., functional or logic issues, resource allocation/dependency issues, and security issues. For

López, J., Kushik, N., Yevtushenko, N. and Zeghlache, D.
Analyzing and Validating Virtual Network Requests.
DOI: 10.5220/0006472304410446
In *Proceedings of the 12th International Conference on Software Technologies (ICSOFT 2017)*, pages 441-446
ISBN: 978-989-758-262-2

the first two groups, we propose the validation techniques based on effective Boolean operations. The third group concerns security issues in user requests and it covers on one hand, the appropriate access control and on the other, potential attacks of the platform. The first problem can be solved through the usage of the security/access control mechanisms for example (Idrees et al., 2015), while the second needs to be considered separately, starting from classical SQL injections and finishing with specific attacks for network virtualization platforms.

Similar work was performed for checking the requests using cloud environments (Huang et al., 2015). However, some of the properties that we have identified cannot be expressed with their configuration predicate language; this partially motivates the current work presented in this paper.

Our main contributions are the identification of the properties to validate, the design and method for integration of a new module into the platform for network services, namely, a user-request-validator for checking logical, resource allocation/dependency and security related request inconsistencies.

We note that since the proposed techniques are generic, platform providers can always add other properties apart from those considered in the paper. Methods and techniques for each group are demonstrated using the user requests written in TOSCA language as it is done for the platform developed and utilized as our case study.

The structure of the paper is as follows. Section 2 has a brief description of the user requests and it also enumerates their potential inconsistencies. Section 3 describes the techniques for the request validation, while Section 4 concludes the paper.

2 ANALYSIS OF ISSUES IN VIRTUAL NETWORK REQUESTS

A tenant request is a structured string, which belongs to the TOSCA language. According to the platform description (Mechtri et al., 2016), each user request can be parsed into three main parts.

One part is concerned about the virtual resources from the virtual devices according to available capacities, ports and other parameters while another part of the request contains a so-called network connectivity topology (*network connectivity graph*). In this graph, Virtual Network Function (VNF) pairs are connected according to the desired topology of neighboring nodes (Fig. 1a). In the third part of the request, Service Function Chains (SFCs) define a sequence (or sequences) of service functions (or VNFs), which are applied to packets or a sequence of packets and should be implemented using the connections of the connectivity graph.

A service function chain defines a (partial) order relation (Nadeau and Quinn, 2015) over a subset of the nodes of the network connectivity graph. Typically, this order relation is given in a graphical form, for example, as depicted in Figure 1b.

```
     NET-2:
             type: tosca.nodes.network.Network
             properties:
                 ip_version: 4
                 cidr:   10.101.0.0/24
                 start_ip: 10.101.0.100
                 end_ip: 10.101.0.150
     NET-3:
             type: tosca.nodes.network.Network
             properties:
                 ip_version: 4
                 cidr:   10.102.0.0/24
                 start_ip: 10.102.0.100
                 end_ip: 10.102.0.150
     Forwarding_path1:
         type: tosca.nodes.nfv.FP
         description: the path (CP11->CP21->CP32)
         properties:
             policy:
         requirements:
             - forwarder: CP11
             - forwarder: CP21
             - forwarder: CP32
groups:
         VNF_FG1:
                     type: tosca.groups.nfv.VNFFG
                     description: VNF forwarding graph
                     properties:
                            vendor:
                            version:
                            connection_point: [ CP11, CP21, CP32 ]
                            dependent_virtual_link: [ VL1 ]
                            constituent_vnfs: [ VNFa, VNFb, VNFe ]
                     members: [ Forwarding_path1 ]
```

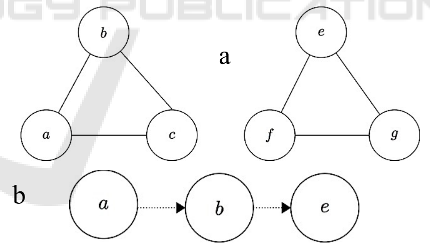


Figure 1: Network connectivity topology graph and network function chain without underlying connectivity.

2.1 Analysis of Properties and Issues

As mentioned above, the network virtualization platform processes virtual network requests expressed by users (or tenants). Based on a vulnerability assessment performed against our network virtualization platform, we have identified three possible groups of inconsistencies, which can occur in the requests: a) functional or logic issues, b) resource/dependency issues, and c) security issues.

A. Functional / Logic Related Inconsistencies Functional or logic related issues arise as the result of misconfigurations. Below we expose some of these issues and the corresponding configuration properties that should hold.

A1. A Chain must not have Loops. A service chain as defined in Section 2 is a partial order, and thus, no loops are allowed in the chain. However, a request compliant with the TOSCA's syntax might contain such issues. Consider the following snippet of code and its corresponding chain representation:

```
VNF_FG1:
          type: tosca.groups.nfv.VNFFG
          description: VNF forwarding graph
          properties:
                vendor:
                version:
                connection_point: [ CP11, CP21, CP11 ]
                dependent_virtual_link: [ VL1 ]
                constituent_vnfs: [ VNFa, VNFb, VNFa ]
          members: [ Forwarding_path1 ]
```

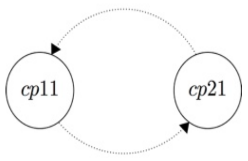


Figure 2: Network function chain with a loop.

As can be seen the chain is formed with a loop. The behavior of the service function chain with a loop might be risky, it can potentially flood the virtual network by infinitely sending the network packets back and forth from / to the different VNFs at the different connection points (CPs).

A2. A Chain can only be implemented if the undelaying connectivity Graph allows it. A service function chain forwards network packets to the respective service functions at each connection point. However, it cannot be possible if the connections points are not logically connected. Consider the snippet of TOSCA code and corresponding network connectivity graph and service function chain described in the Fig. 1.

The service function chain logically connects packets coming from a node at the network called 'NET-2' to a node at the network called 'NET-3'. The problem is that forwarding traffic from one to the other violates the virtual network isolation. If the traffic is not forwarded, then the chain is not respected.

A3. Some Chains must have a "Complementary" reverse Chain. The majority of communication protocols rely on some form of bilateral communication, even if just for acknowledging the reception of data. For that reason, having a reverse chain for each chain at least from the 'last' to the 'first' connectivity point might be desirable. In the previous two examples, none of the chains depicts a reverse chain. These declarations might be candidates to trigger warnings. Considering the case in Fig. 1, if two chains CP11→CP21 and CP21→CP11 are implemented, this request is considered as valid.

B. Resource / Dependency Issues

Network service functions are implemented as running software which has resource requirements. Moreover, each service function runs on a computational device, and the service intrinsically impose restrictions on the devices executing it.

B1. Restrictions on the VNF Neighboring Connections. Certain VNFs might require specific configurations. For example, if a VNF is of the type 'HTTP Cache' it is necessary that such VNF is connected to an 'HTTP Server' VNF. The reason is that any HTTP cache needs an origin HTTP server from which it takes the data to cache. Furthermore, they must be connected through a service function chain.

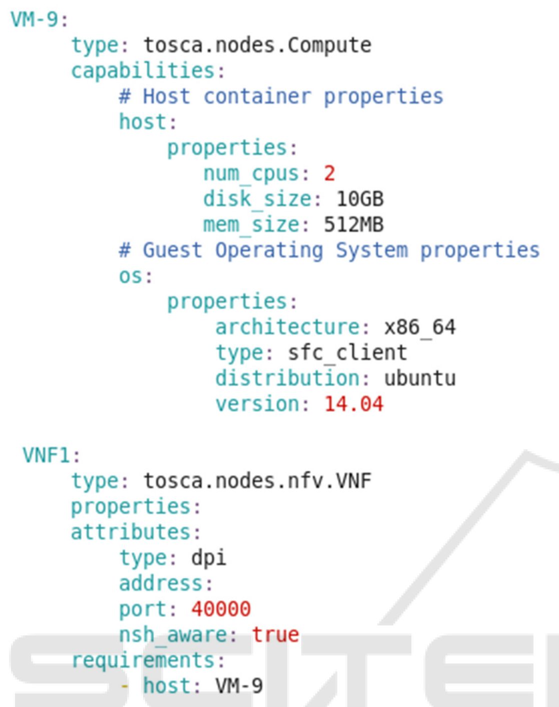
B2. Restrictions on the Resources used by VNFs. A VNF might be known to be resource intensive. As an example, consider a deep packet inspection (DPI). The corresponding VNF requires fast processing of the data packets if the DPI is performed on-line. Therefore, the associated Virtual Machine (VM) needs to have an appropriate number of CPUs, for example least 4 CPUs. Fig. 3 depicts an invalid configuration for the DPI VNF.

B3. Resource Capacity Dependencies. Dependencies across different properties of the (TOSCA) nodes might be implied. For instance, the outbound bandwidth of a node in a SFC must be less or equal than the inbound node which follows it. Another interesting issue is the node resource interdependency, i.e., the dependencies that each node imposes according to a specific declared resource. For example, consider a VNF which is CPU intensive with enough CPU capacity. However, if the inbound bandwidth is very high the possessed CPU is not sufficient. There exist a large number of resource implications inside a user specification. However, due to space constraints in this publication, we list a few of them, and omit several others, which might be more trivial. For example, the implications of a unique resource assignation as a port or an IP address.

C. Security Related Issues

As any application that processes complex user requests, the virtualization platform might be potentially vulnerable to certain inputs. In the following, we analyze some possibilities.

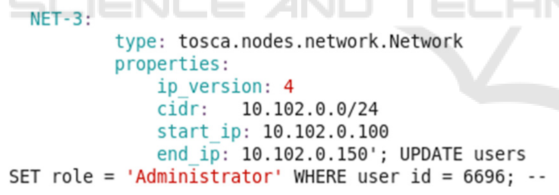
C1. Code Injection Issues. As the request is passed via a text format, the platform might use certain values found in the original text of the request to process the request itself. For example, it could use the value found in the IP address range for setting some value(s) in the database. In Fig. 4, we illustrate an SQL injection attack in the request.



```
VM-9:
     type: tosca.nodes.Compute
     capabilities:
         # Host container properties
         host:
             properties:
                num_cpus: 2
                disk_size: 10GB
                mem_size: 512MB
         # Guest Operating System properties
         os:
             properties:
                 architecture: x86_64
                 type: sfc_client
                 distribution: ubuntu
                 version: 14.04

 VNF1:
     type: tosca.nodes.nfv.VNF
     properties:
     attributes:
         type: dpi
         address:
         port: 40000
         nsh_aware: true
     requirements:
       - host: VM-9
```

Figure 3: Invalid configuration of the DPI VNF.



```
  NET-3:
          type: tosca.nodes.network.Network
          properties:
              ip_version: 4
              cidr:   10.102.0.0/24
              start_ip: 10.102.0.100
              end_ip: 10.102.0.150'; UPDATE users
SET role = 'Administrator' WHERE user id = 6696; --
```

Figure 4: SQLi attack on TOSCA specification.

Other types of attacks are possible as well. For instance, an attacker might inject a cross-site scripting code to steal the cookies of the platform administrator in order to obtain super user privileges. Given the fact that the platform might use open source software, verifying the absence of such attacks in the requests is crucial.

C2. Denial of Service (DoS) Issues. Generally speaking, any platform which is open for requests is susceptible to DoS attacks. Not considering all the network DoS or delayed data sending DoS, which are assumed to be handled by other system components, there exist certain types of DoS attacks, for which a processed request might be used. For that reason, we mind their implications. In fact, a sequence of requests which are similar in a short period of time might prevent the whole system to execute any further actions. Furthermore, the requests can also be not valid, or partially non-valid.

3 REQUEST VALIDATION

Different mathematical models can be applied for verifying the request consistency. For checking the properties of type A, we use Boolean matrices for the graph representation and operators over them. A network connectivity topology is an undirected graph $G = (V, E)$ where the set $V = \{VNF_1, \ldots, VNF_k\}$ of nodes represents network devices while edges of the graph (the set E) are pairs (VNF_i, NNF_j), $VNF_i, VNF_j \in V$; edges represent connections between two nodes. As mentioned above, in virtual networks a chain is considered as a (partial) order (Fig. 1b). A *Service Function Chain* (SFC) defines a strict (partial) order relation on the set of service functions (or VNFs); this relation can be represented as a list of corresponding VNF pairs or as a directed graph. Given an SFC over the set V of VNFs, VNF_j is a *direct successor* of $VNF_{j\text{-}1}$ if the relation has a pair $(VNF_{j\text{-}1}, VNF_j)$, written $VNF_{j\text{-}1} \rightarrow VNF_j$, while VNF_j is a *successor* of VNF_t, written $VNF_t \mapsto VNF_j$, if the order relation has pairs $VNF_t \rightarrow VNF_{t+1} \rightarrow \ldots \rightarrow VNF_j$. An SFC is an acyclic directed graph and thus, Boolean adjacency matrices can be used for representing the network connectivity graph as well as the set of SFCs. If the matrix **G** represents a directed or an undirected graph then the matrix $\mathbf{G}^n$ i.e., the product of n matrices **G**, represents the set of nodes which are connected via a path of length n.

Checking the A1 Property. To check that a given set of VNF pairs is a partial order, i.e., an SFC, the relation is represented by a Boolean matrix **F** of a corresponding directed graph and then the problem reduces to checking if the graph has no cycles. The matrix $\mathbf{F}^n$ is derived up to appropriate n. Once some diagonal item of $\mathbf{F}^n$ becomes non-zero, the graph is not acyclic. Otherwise, the given set represents a partial order relation.

As an example, consider the request in Fig. 1 and a corresponding Boolean matrix $\mathbf{F} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. The matrix $\mathbf{F}^2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and $f_{11} = 1$, i.e., the requested SFC is not a partial order.

Checking the A2 Property. If the network connectivity graph and a requested SFC are represented as Boolean matrices **G** and **F** correspondingly, then it is enough to check that each row of **F** is contained in the corresponding row of **G**. If the SFC is represented as a list of pairs then it is enough to check that for each pair $VNF_i \rightarrow VNF_j$. When reachability problems for the connectivity graph *G* are investigated it is natural to consider the Boolean matrix **F** while representing a chain as an adjacency list. Then for each pair $VNF_i \rightarrow VNF_j$ of the chain where we check whether the (*i*, *j*) cell of **G** equals 1. If not then the chain cannot be implemented by the connectivity graph *G*.

Checking the A3 Property. As mentioned above, each VNF must respect possible predecessors and successors and such example is given when describing the A3 property. If the list of direct successors is given as a list of pairs then it is enough to check that no SFC contains such pairs. If any successors are considered then it is better to represent an SFC as a Boolean matrix **F** and check whether a 'forbidden' pair induces '1' at the corresponding cell of the matrix $\mathbf{F}^n$.

For checking the properties of type B, we use systems of logical implications which can be represented as a CNF that is the product of disjunctive clauses. Given a request, if CNF is not equal to '1' then the well-known SAT problem can be used for verifying an inconsistent request.

Checking the B1 Property. The property of SFC to have a reverse chain can be described by a logical implication that can be later converted to a CNF. For example, if there is an SFC $\alpha = VNF_1 \rightarrow VNF_2 \rightarrow VNF_3$ for which there should exist a reverse chain such that $VNF_3 \mapsto VNF_1$ then there are logic expressions for the following implication: $\boldsymbol{if}\ (VNF_1 \rightarrow VNF_2) \wedge (VNF_2 \rightarrow VNF_3)\ \boldsymbol{then}\ (VNF_3 \mapsto VNF_1)$. The above expression can be represented by the following CNF: $\overline{(VNF_1 \rightarrow VNF_2)} \vee \overline{(VNF_2 \rightarrow VNF_3)} \vee (VNF_3 \mapsto VNF_1)$ (**a**).
If the CNF is not equal to '1" then there is no chain such that $(VNF_3 \mapsto VNF_1)$.

Checking the B2 Property. The property states that certain VNFs might have resource limitations. Assume that the example shown in Fig. 3 applies for VNFs of the 'dpi' and 'fw' types (as the FW type of VNF is also resource consuming), then the associated logic expression (implication) is the following: $\boldsymbol{if}\ (VNF_i.type = "dpi") \vee (VNF_i.type = "fw")\ \boldsymbol{then}\ VNF_i.cpu \geq 4$.

The above expression can be represented by the following CNF:

$$\left(\overline{VNF_i.type = "dpi"} \vee (VNF_i.cpu \geq 4)\right) \wedge \left(\overline{VNF_i.type = "fw"} \vee (VNF_i.cpu \geq 4)\right) \ (\mathbf{b}).$$

Checking the B3 Property. There are two examples stated for property B3. The first example states that in an SFC, the outbound bandwidth of a given VNF should be less or equal than the inbound bandwidth of the subsequent VNF. This property can be described by the following implication: $\boldsymbol{if}\ (VNF_i \rightarrow VNF_j)\ \boldsymbol{then}\ VNF_j.inB \geq VNF_i.inB$.

The above expression can be represented by the following CNF: $\overline{(VNF_i \rightarrow VNF_j)} \vee (VNF_j.inB \geq VNF_i.inB\ )$ (**c**).

The second example for B3 states that certain characteristics of the virtual machines are inter-dependent. For example, the CPU depends on the inbound bandwidth. For the scope of this paper, we assume the dependency is linear and the lower bound on the minimal demanding bandwidth is a constant. However, thorough investigation of this correlation is necessary. Assume that the lower bound is 100MB/s. In this case, the associated logical implication can be described by the following implication: $\boldsymbol{if}\ (VNF_i.inB \geq 100MB/s)\ \boldsymbol{then}\ VNF_i.cpu > 2 * VNF_i.inB$.

The above expression can be represented by the following CNF: $\overline{(VNF_i.inB \geq 100MB/s)} \vee (VNF_i.cpu > 2 * VNF_i.inB\ )$ (**d**).

In this paper, we do not discuss trivial inconsistencies of a tenant, e.g., inconsistent requested capacities (for example, requested memory exceeds the maximal physical limits), etc. All such inconsistencies can be checked simply enough using arithmetic rules (mostly inequalities) such as $VNF_i.mem \leq MEM\_MAX$.

There can be more examples of logic implications when consistency rules are extended for applied requests. However, as it can be seen there is a system of logic implications which can be written as a CNF. This CNF is in fact the conjunction of all previously depicted CNFs, i.e., $a \wedge b \wedge c \wedge d$.

According to the CNF, a request is *resource consistent* if the CNF result equals to '1'. Using CNFs opens more possibilities. For example, if the CNF equals 0 for a given request we could solve a corresponding SAT problem in order to find a solution that is the closest to the request data.

For checking properties of the C group, we propose the use of different methods, which are known to be good for detecting the corresponding security threats.

Checking the C1 Property. In order to check that the request is safe with respect to code injection attacks, different approaches have been proposed. Nonetheless, among all these methods, anomaly detection using supervised machine learning techniques have proven to be highly effective (Watson et al., 2016). For example, a training set with negative labels for strings containing ill-formatted strings could report a SQL injection as an invalid request. Another possibility is to avoid the verification and directly transform the input into a 'safe' one through the string analysis. In this case, the request can be *sanitized* in order to be consistent (Alkhalaf, 2014).

Checking the C2 Property. In order to avoid the DoS attacks, many approaches have been proposed as well (Zargar et al., 2013). Depending on the type of an application that processes the request, different approaches have been proven to be effective. In our studies, the majority of virtualization platform orchestrators are implemented in any form of a web service or web application. For that reason, such platforms are susceptible to a number of DoS including the attacks that exploit slow request/response or fragmentation attacks. One possibility to avoid these threats is to make use of anomaly detection methods applied to the users' behavior.

We note that the properties of types A-C described above form just a small subset of the issues that can appear in user requests. However, the scalable solutions discussed above can be applied to other types of inconsistencies.

4 CONCLUSIONS

In this paper, we addressed the problem of the verification and validation of user requests for systems providing network services.

We discussed the possibilities of checking three types of request issues, namely functional/logical issues, resource allocation or parameter dependency issues, and finally security issues. We also proposed a number of scalable techniques for solving the problems listed above and illustrated these techniques by a number of examples of user requests.

As for the future work, we first plan to implement the proposed request-validator solution and then perform experimental evaluation in order to prove its effectiveness. As one of existing platforms providing virtual networks and service function chains has been developed in our previous works, we plan to use it as a case study.

Finally, we plan to study other non-functional issues that can be added to the verification/validation process of the user request.

ACKNOWLEDGEMENTS

The results in this work were partially funded by the Celtic-Plus European project SENDATE, ID C2015/3-1; French National project CARP (FUI 19); Bilateral contracts with Orange Labs; Russian Science Foundation (RSF), project № 16-49-03012.

REFERENCES

European Telecommunications Standards Institute (ETSI), 2013. Network Functions Virtualisation (NFV); Use Cases, NFV-MAN V1.1.1, ETSI Standard.

Mechtri, M., Benyahia, I. G., Zeghlache, D., 2016. Agile service manager for 5G, in the proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), Istanbul, Turkey, pp. 1285-1290.

Nadeau, T., Quinn, P., 2015. Problem Statement for Service Function Chaining, Internet Engineering Task Force (IETF) Request for Comments (RFC) 7498.

Palma, D., Rutkowski, M., Spatzier, T, 2016. TOSCA Simple Profile in YAML Version 1.0. OASIS Committee Specification 01.

Huang, P., Bolosky, W. J., Singh, A., Zhou, Y., 2015. ConfValley: a systematic configuration validation framework for cloud services, in the proceedings of the Tenth European Conference on Computer Systems (EuroSys '15). New York, USA, pp. 19:1-19:16.

Watson, M. R., Shirazi, N.-u.-h., Marnerides, A. K., Mauthe, A., Hutchison, D., 2016. Malware Detection in Cloud Computing Infrastructures, IEEE Transactions on Dependable and Secure Computing, vol. 13, no. 2, pp. 192–205.

Zargar, S. T., Joshi, J. Tipper, D., 2013. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks, IEEE Communications Surveys & Tutorials, vol. 15, no. 4, pp. 2046-2069.

Idrees, M. S., Ayed, S., Cuppens-Boulahia, N. and Cuppens, F., 2015. Dynamic Security Policies Enforcement and Adaptation Using Aspects, in the preceedings of the IEEE Trustcom/BigDataSE/ISPA, pp. 1374-1379.

Alkhalaf, M. A., 2014. Automatic Detection and Repair of Input Validation and Sanitization Bugs. PhD thesis, University of California, Santa Barbara.