J Reliable Intell Environ (2015) 1:123–146
DOI 10.1007/s40860-015-0008-0

ORIGINAL ARTICLE

Modelling cyber security for software-defined networks those grow strong when exposed to threats

Analysis and propositions

Usama Ahmed[1,2] · Imran Raza[1] · Syed Asad Hussain[1] · Amjad Ali[1] · Muddesar Iqbal[3] · Xinheng Wang[3]

Received: 19 August 2015 / Accepted: 15 October 2015 / Published online: 24 November 2015

Abstract With each passing day, the information and communication technologies are evolving with more and more information shared across the globe using the internet superhighway. The threats to information, while connected to the cyber world are getting more targeted, voluminous, and sophisticated requiring new antifragile and resilient network security mechanisms. Whether the information is being processed in the application, in transit within the network or residing in the storage, it is equally susceptible to attack at every level of abstraction and cannot be handled in isolation as the case has been with conventional security mechanisms. The advent of Software-Defined Networks (SDN) has given a new outlook to information protection, where the network can aid in the design of a system that is secure and dependable in case of cyber threats. The nature of SDN, mainly its programmability and centrality of network information and control has led us to think of security in an antifragile perspective. Our networks can now thrive and grow stronger when they are exposed to volatility by overwhelming cyber threats. However, SDN infrastructure itself is susceptible to severe threats that may mutilate the provision of its usability as security provider. Both these perspectives of "Security with SDN" and "Security for SDN" have invited research and innovations, yet both these approaches remain disintegrated, failing to support each other. The contribution of this paper is threefold, with first reviewing the current state of the art work for both perspectives of SDN security. Second, it advocates the necessity and introduces a novel approach of antifragile cyber security within SDN paradigm and finally it proposes a unified model for integrating both approaches of "Security with SDN" and "Security for SDN" to achieve the overall objective of protecting our information from cyber threats in this globally connected internetwork.

Keywords Modelling · Antifragility · Cyber security · SDN

✉ Muddesar Iqbal
director.oric@gmail.com

Usama Ahmed
fa14-pcs-004@ciitlahore.edu.pk; usamaahmed@gcuf.edu.pk

Imran Raza
iraza@ciitlahore.edu.pk

Syed Asad Hussain
asadhussain@ciitlahore.edu.pk

Amjad Ali
amjad.ali@ciitlahore.edu.pk

1 Department of Computer Science, Communication and Networks Research Centre, COMSATS Institute of Information Technology, Lahore, Pakistan

2 Department of Software Engineering, Government College University, Faisalabad, Pakistan

3 Pak-UK Institute of Innovative Technologies for Disaster Management, University of Gujrat, Gujrat, Pakistan

1 Introduction

Taleb defines antifragility as a nonlinear response, stating that "Simply, antifragility is defined as a convex response to a stressor or source of harm, leading to a positive sensitivity to increase in volatility" [1]. The basic properties of a secure system demand its protections to grow stronger throughout its entire lifecycle to fight against the overwhelming security threats. This is especially the case when malicious attacks and unintentional damages to data, assets, and communication transactions are always imminent from inside as well as outside the network.

A typical network running commercial applications accessed by users from across the internet fears far more risks than its intranet counterpart. A traditional cyber attack to any organization's information resources has mostly consisted of breaking into the network by exploiting vulnerabilities in the internet facing systems. Using these victims machine, the intruder has to really work hard to gain access to internal network and its diverse segments installed with a variety of middleboxes. These middleboxes may have their own set of vulnerabilities requiring tremendous effort, skill and hard work for exploitation.

The nature of SDN [2], mainly its central control and programmability has transformed the entire paradigm of enterprise security inviting a divided opinion. SDN's nature, as one opinion, is regarded as a point of failure and the weakest link in the chain of network security measures offering a highly significant and lucrative option for intruders. However, in the other opinion, this very nature of SDN is the strongest aspiration for SDN adoption due to its ease of control and jurisdiction for enabling and managing security functions in a network. This later opinion influenced by the SDN's resilient and antifragile nature has made us think differently for enterprise security from the trending cyber threats.

A new perspective of cyber antifragility can be thought of as the ability to deal with an attack, return to normal operation with minimal damages and with enhanced capability to thwart the future challenges of known or unknown types.

A basic SDN architecture, offering programmability and control logic centralization can be seen from Fig. 1. The infrastructure layer or the data plane consists of network elements merely capable of forwarding data to the selected destinations. The infrastructure layer is managed by the control layer which offers network services and control logic centralization. Application layer sits on top of control layer and offers user-driven application and services. SDN's control layer mediates between the networking requirements that arise from the application layer and translates them into instructions for the network elements in the infrastructure layer. However, these capabilities have introduced new faults and vulnerabilities that are easily exploitable with new avenues of threats opened by the SDN framework.

To ensure the security of the system throughout life time, it is required to introduce a functional method of identifying challenges associated with security of the application, control and data plane along with their associated interfaces. An SDN-enabled reference network offering state of the art capabilities and services to its users is elaborated in Fig. 2. The application layer consists of a range of user-specific applications that interact with underlying pool of shared resources. This process is arbitrated through one or more controllers and network elements. The said network provides services and management capabilities to users inside the network and across the internet, hence requiring protection from imminent threats.

In contrast to the work above, the security of SDN is evaluated in [5] against the conventional networks across a set of five key parameters of security namely the *Confidentiality, Authenticity, Integrity, Availability and Consistency*. It is established that the threats to the SDN infrastructure are not new and can be tackled in more or less the same way as of the conventional networks. The need of the hour is to reap the benefits of SDN design that outweigh the security problems introduced by SDN.

This wide disparity in SDN security is a driving factor to consider an overall, simplistic, unified approach towards achieving both elementary principles of "Security with SDN" and "Security for SDN" as shown in Fig. 3. The overall aim is to provide a continuously evolving system-wide protection that wraps up the application level and underlying network level security requirements. This paper presents a novel antifragile cyber security model that encompasses the

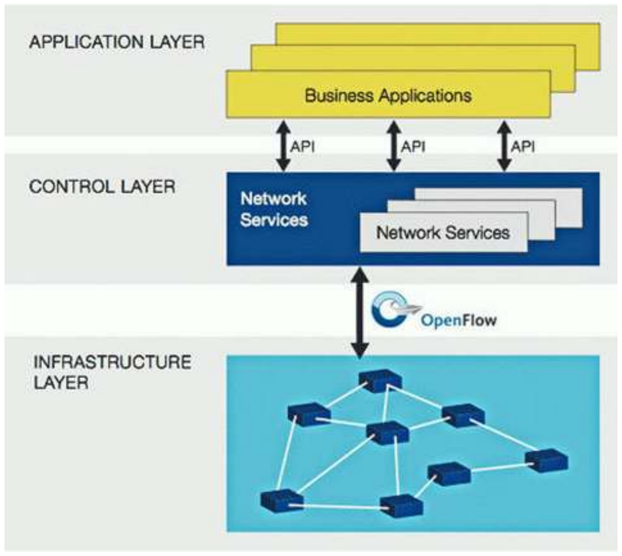


Fig. 1 Basic SDN architecture

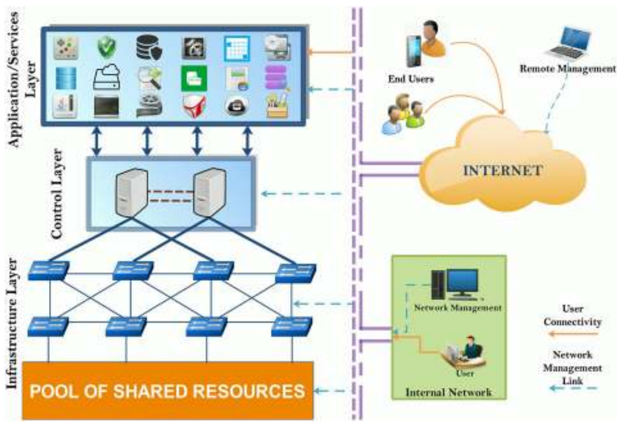


Fig. 2 SDN-enabled network offering services to end user

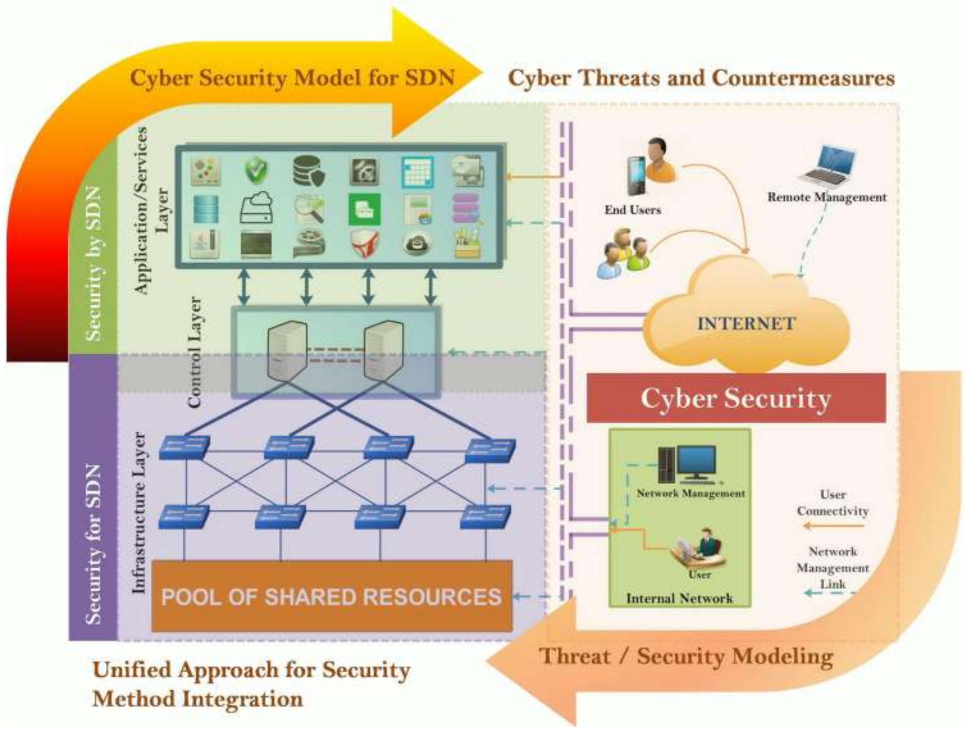


Fig. 3 Cyber security model for SDN

known threats to the SDN-enabled systems, their outcomes, possible locations and countermeasures offered by SDN or with addition of external security devices.

The rest of the paper is distributed as follows. Sections 2 and 3 pertain to a survey of recent advancements in both perspectives of SDN security and Sect. 4 highlights the security and threat modelling approaches. Section 5 presents our proposed Antifragile Cyber Security Model and debates the idea and need for security that is beyond resilience in the realm of SDN. Section 6 concludes the discussion and Sect. 7 presents our ideas about the way forward for the proposed model.

2 Survey of security enhancement using SDN framework

The idea of using SDN framework for enabling security in enterprise environment is a widely held belief. The argument is based on the exploitation of SDN architecture to provide enterprise level security. This section describes the state of the art work in security with SDN, highlighting the enhancements using SDN framework.

2.1 Monitoring systems and middleboxes

The integration of middleboxes in a network based on SDN leverages the centrally controlled nature of programmable networks to redirect selected traffic through them. The authors in [6] suggest SLICK, which is a central control plane responsible for installing and migrating functions onto custom middleboxes in the network. Applications direct SLICK controller for routing particular kind of traffic through middleboxes. There are three main components of SLICK architecture namely (1) Control Plane protocol, (2) The Programming Model and (3) The SLICK Controller. SLICK's primary goal is to support a network operator to easily implement policies across all the middleboxes in the network. This results in efficient deployment of middleboxes with lowest possible path overhead for traffic.

A modification of middleboxes is proposed as FlowTags in [7] which interact with SDN controller through FlowTags API. Flow tags are embedded in packet headers to provide flow tracking. The dynamic and traffic-dependent modifications by middleboxes make it difficult to establish the correctness of network-wide policy enforcement (e.g. access control, accounting, and performance diagnostics). The solution is a new “southbound” controller–middlebox interface that enables SDN controllers to configure the flow tagging capability, and the support needed from middleboxes to implement FlowTags-related functions. The flow tracking capability is introduced to ensure consistent policy enforcement in the presence of dynamic traffic modifications.

FlowTags has three key dimensions highlighted in Fig. 4:

1. FlowTags-enhanced middleboxes that read incoming packet's existing Tags while processing it and may also add new Tags based on the context.

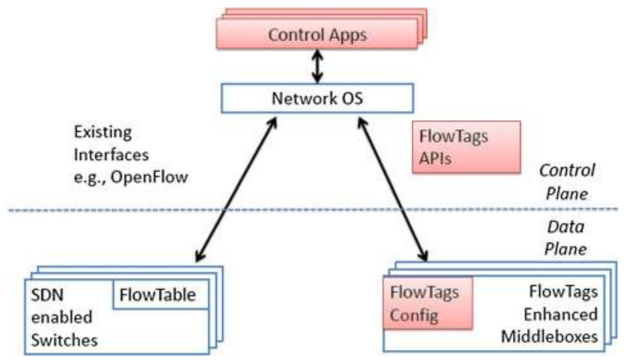


Fig. 4 Flowtag architecture [7]

2. FlowTags APIs between the FlowTags-enhanced middleboxes and controller.
3. Control applications that configure the tagging behaviour of the middleboxes and switches, and that also organize Tags to support policy enforcement and verification.

Qazi et al. [8] describe SIMPLE (Software-defIned Middlebox PoLicy Enforcement) which is an SDN-based policy enforcement layer that translates a high-level middlebox policy into an efficient and load balanced data plane configuration. It then steers traffic through the desired sequence of middleboxes without making any modification to SDN architecture or the middleboxes. Figure 5 gives an overview of the SIMPLE architecture showing the inputs, the modules' interactions, and the interfaces to the data plane. SIMPLE controller has three basic modules described as below.

1. *ResMgr module* which is responsible for implementing policy requirements. The input to this module is the network's traffic matrix, policy requirements and topology. This module is also responsible for optimally balancing the load across middleboxes by taking into account middlebox and switch constraints.
2. *DynHandler module* using a lightweight payload similarity algorithm, correlates the incoming and outgoing connections of middleboxes and provides these mappings to the RuleGen module.
3. *The RuleGen module* takes the output of both the ResMgr and DynHandler modules and generates data plane configurations for routing the traffic across the appropriate sequence of middleboxes to their destination.

A solution for network monitoring called Open Security Auditing and Flow Examination (OpenSAFE) is presented [9]. Routing of traffic in a robust and high performance manner for network analysis is discussed along with A Language for Arbitrary Route Management for Security (ALARMS) policy language. ALARMS is used for simplified management of monitoring devices within the network. Moreover,

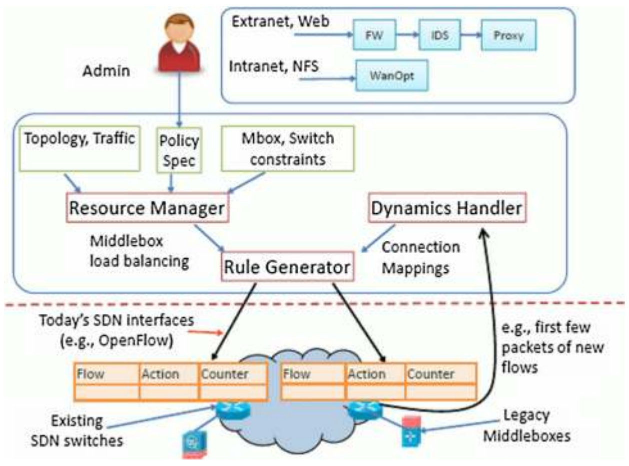


Fig. 5 SIMPLE Overview: Using SDN to manage middleboxes [8]

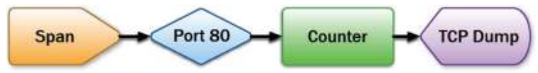


Fig. 6 Monitoring path abstraction [9]

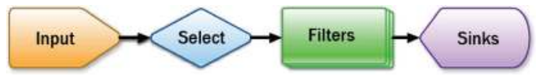


Fig. 7 A basic logical monitoring path [9]

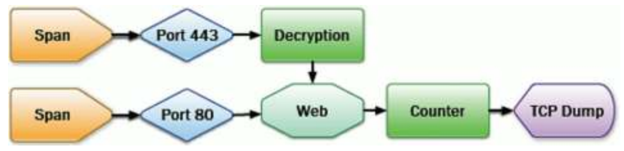


Fig. 8 A logical path with a waypoint [9]

OpenSAFE consists of a set of design abstractions for thinking about the flow of network traffic and an OpenFlow component that implements the policy.

The basic abstraction of OpenSAFE design is the *path* which is the route that a particular traffic will take. Other levels of design abstraction consist of *input, selection, filter and sink.* Each *path* starts with *input*, on which optional selection criteria are applied and afterwards the traffic is routed through any or none of the *filters*, finally ending up at one or more *sinks.*

This concept is elaborated in Fig. 6 as abstraction and as logical monitoring path in Fig. 7.

Another abstraction as shown in Fig. 8, that may be used to aggregate policy rules and reduced repetition is the "virtual destination" called the *waypoint.* The abstraction of OpenSAFE is translated into the ALARMS policy language where all components are given names and definition according to their role.

Starting with the *inputs* and *sinks*, which are simply the OpenFlow switch ports, and ending with *waypoint*, the definition according to ALARMS language is given by

```
Input span = of:0
Sink tcpdump = of:1
filter to counter = of:2;
filter from counter = of:3;
select http = tp src: 80 || tp dst: 80;
waypoint web;
```

The abstraction shown in Fig. 8 can be translated into policy language as

```
    span[http] -> web;
    span[https] -> decrypt -> web;
    web -> counter -> tcpdump;
```

Another feature of OpenSAFE is the distribution of traffic between several components to support, for instance, load balancing. The distribution rules are applied on per-flow basis, and can be one of four namely, *ALL, RR, ANY or HASH*. The traffic is duplicated to all components in case of ALL rules, round-robin in case of *RR* and random in case of *ANY*. The *HASH* rule is somewhat different and depends on the name of hash function to determine the destination of the traffic. OpenSAFE is implemented using OpenFlow v0.9.8 and NOX v0.6 with prototype built in Python and suggests an overall ease in managing and monitoring network traffic in large- scale networks.

The authors in [10] present 'CloudWatcher' in which the traffic in cloud networks is detoured to pass between monitoring systems using a simple policy script. The solution not only ensures that all necessary traffic is passed through some security device for monitoring but also provides a simple language for writing policy scripts to manage the traffic.

CloudWatcher works as an application on top of any OpenFlow controller and starts working by registering security devices using *SLI-registration script* knowing their *device ID, device type, location, installation mode and supported functions.* The next step is the creation of security policies, which consist of Flow condition representing the flow to be investigated and device set which shows the security devices participating in the investigation. An example SLI-policy script for monitoring traffic between two hosts A and B can be given by $\{A :^* \rightarrow\ B :^*, \{Device\_ID\}\}, where\ A :^*$ and $B\ :^*$ depicts all traffic originating from two hosts. These hosts are identified by their *IP addresses* and *Device_ID* is any number assigned to the monitoring device.

Whenever CloudWatcher finds network packets meeting a specified policy, it routes these packets to satisfy given security requirements. CoudWatcher also ensures that the routes used by these packets are always optimized. Path optimization cannot be handled by conventional routing protocol in the case of added security devices in the path. For the same reason, the shortest path problem is formulated as minimum cost flow problem as in Eq. (1).

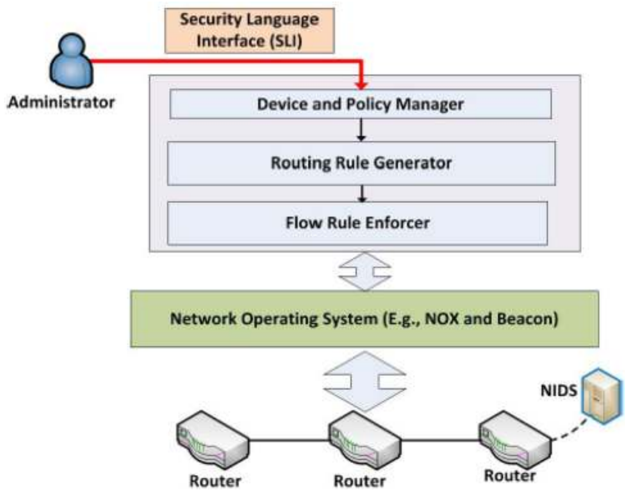

Security Language Interface (SLI)
Administrator
Device and Policy Manager
Routing Rule Generator
Flow Rule Enforcer
Network Operating System (E.g., NOX and Beacon)
NIDS
Router
Router
Router

Fig. 9 CloudWatcher overall design [10]

$$\begin{aligned} &\min \sum c_{i,j} x_{i,j} \\ &\text{s.t} \sum_{j=1}^{n} x_{i,j} - \sum_{k=1}^{n} x_{k,i} = b_i \quad \text{for} \quad i = 1, 2, \ldots n \\ &x_{i,j} \geq 0 \quad \text{for} \quad i, j = 1, 2, \ldots, n \end{aligned} \qquad (1)$$

where $x_{i,j}$ represents the amount of data sent along the link from node i to node j, b_i means available supply at node i, for $b_i \leq 0$ there is a required demand at node i, $c_{i,j}$ is the unit cost for flow between node i and j. A set of four algorithms is used to find the shortest paths inclusive of security devices, in different scenarios. Three of these algorithms are used to find paths that include passive security devices and one is used to find path with inline security device.

The proposed framework of CloudWatcher is implemented as an application on top of NOX controller [11], using python. The implementation consists of three main modules as shown in Fig. 9 and described below,

1. *Device and Policy Manager* that maintains a device table containing information of each security device, and a policy table containing information on each security policy.
2. *Routing Rule Generator* which initially sends queries to network switches or routers for network discovery and afterwards generates a network topology as a graph structure. It periodically sends query through NOX APIs to estimate the cost of links within the network. A modified 'Dijkstra' algorithm [12] is used to find the shortest path between two nodes.
3. *Flow Rule Enforcer* parses routing rules or response strategies, and it translates them into flow rules for OpenFlow and sends them to routers or switches.

However, the proposed framework of CoudWatcher may suffer from degradation in case of high network loads, or in case

Table 1 MTTV Table [13]

Switch name	IP	MAC	Threshold ()
S1	192.168.10.101	05-16-DC-59-C2-34	3
S2	192.168.10.102	FF-EC-DC-12-43-33	4
......	......	......	......
Sn	192.168.10.187	DC-FE-32-29-28-37	3

there are security devices that are disconnected from each other thus jeopardizing the path generation algorithm.

The integration of SDN-based middleboxes is fully justifiable in terms of benefits they offer. However, their appropriate placement in the network along with the performance penalty that can be tolerated by an additional link are few questions still unanswered and needs further investigation into the idea.

2.2 SDN for security as a service

This section reviews the state of the art work in which SDN is used for providing security as a service in the basic networks or in the cloud. The research work can be broadly categorized as techniques for (1) Spoofing Detection, (2) Scanning and Anomaly Detection (3) DoS/DDoS Detection and Mitigation (4) Forensic Analysis and (5) SDN-enabled Security Framework.

2.2.1 Spoofing detection techniques

The very basic of any attack to a resource is to hide or fake the identity of the attacker or the resources used in generating the attack. This mechanism of spoofing can be thwarted using the SDN framework. A solution to the problem of device or MAC spoofing resulting in the addition of malicious switches within the SDN is discussed in [13]. A rogue switch can mislead the controller to make incorrect decisions which may jeopardize the performance of overall network. Two schemes are presented for detecting malicious switches, one is based on Threshold Value Control (TVC), and the other argues on installing a third party server in the network to facilitate user feedback for the network.

A table containing Maximum Traffic-flows Threshold Value (MTTV) is maintained, for TVC, to store the maximum traffic flows of each switch in Open Flow networks as shown in Table 1. The controller finds out the threshold value of each switch's maximum traffic flows by learning from its working history. This owes to the fact that if any rogue switch wants to lure controller to pass traffic through it, this is only possible by pretending to have more bandwidth than actual. The controller can, however, refer to the MTTV table to identify the suspicion on a rogue switch which can then further be investigated. The method of investigation is, however, not elaborated in their proposed work.

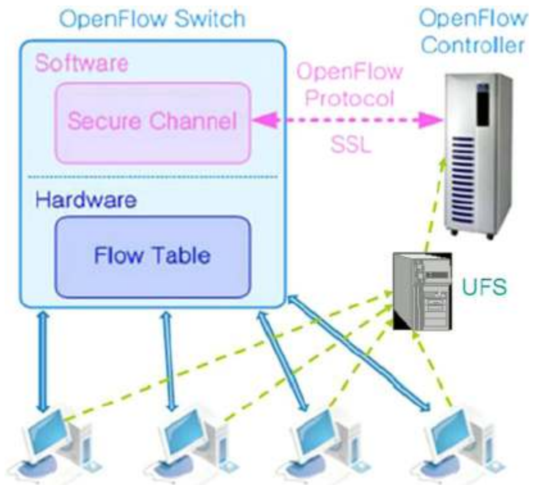


Fig. 10 UFS placement in SDN [13]

The proposed scheme becomes complimented by a second proposal which is presented as an aid to the mentioned scheme. A User Feedback Server (UFS) considered as a trusted third party server is placed inside the network as shown in Fig. 10.

The UFS is used to collect user experiences whenever users are having trouble to connect to the network or there is no response to the traffic they originated. The feedback is sent to the UFS which collects all feedbacks for a certain period of time and send them to the controller. In this way, the controller can find some suspicious paths and suspicious switches within the network.

The mentioned approaches in [13] are fairly simple but their significance is questionable. The proposed work is not supported by any practical implementation and hence its impact is not well justified. Moreover, no comparison is made to any other approaches available in literature. Performance and traffic overheads, false positives and many other parameters should have to be calculated for a healthy impact.

The authors in [14] describe an incrementally deployable anti-spoofing mechanism for SDNs named as BASE (BGP-based Anti-spoofing Extension). An incrementally deployable protocol, as argued, should have three properties: initial benefits for early adopters, incremental benefits for subsequent adopters, and effectiveness under partial deployment. Three techniques lie at the heart of BASE, namely (1) Mes-

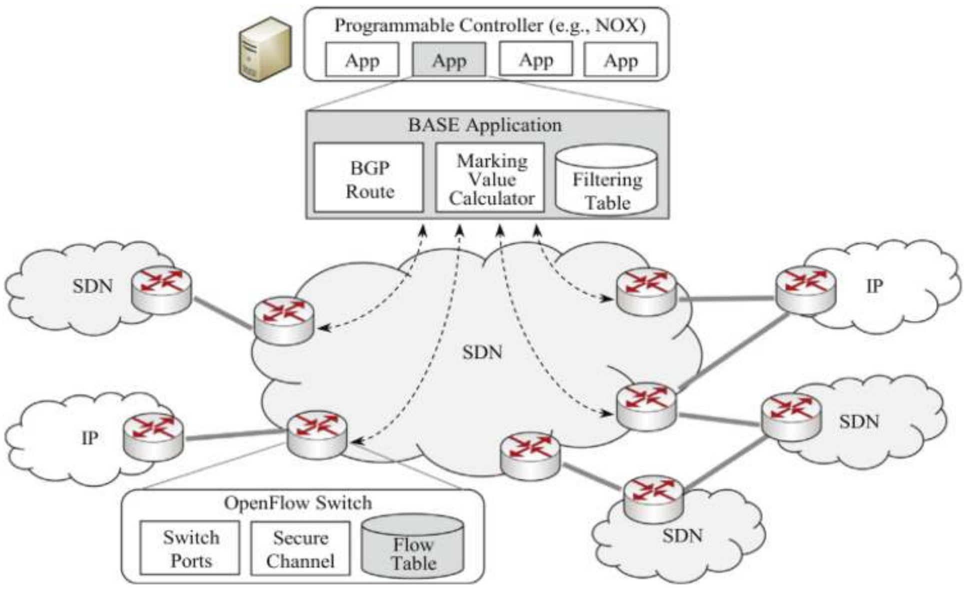


Fig. 11 BASE architecture [14]

sage Authentication Code (MAC), (2) one-way hash chains and (3) packet marking. MAC and one-way hash chains are used for generating a cryptographically unique value for a filter node, whereas packet marking is employed for storing and delivering the value to destinations. The BASE architecture as shown in Fig. 11 inherits and combines the features from Pi [15] and DPF [16] as having per-packet deterministic packet marking inherited from former and overloading a routing protocol BGP inherited from the later to propagate marking information.

However, BASE extends concept of Distributed Packet Filtering (DPF) with cryptographic packet marking, making it more suitable for verifying path correctness. The BASE mechanism is divided into four phases namely, (1) Distribution of marking values, (2) Filter invocation, (3) Packet marking and filtering and (4) Filter revocation. The approach not only works well on the SDN architecture in a high network load, but also recovers network connectivity automatically by filtering the spoofed packets. BASE has a very small overhead during the distribution phase as the markings are piggybacked with BGP update messages. The invocation and revocation phases incur negligible overhead, since only a single BGP update is used to initiate each start or stop signal.

Another lightweight and efficient framework for route-based IP spoofing filtering, named Software dEfined Filtering Architecture (SEFA), is proposed in [17]. SEFA provides a collective view of the network and decouples the filtering rule generation from network devices. It is a hybrid architecture in which filtering is managed by OpenFlow and forwarding is still handled by routing protocols. The design of SEFA enables to collect and build the overall view of address assignments and routing within the network. It presents this view to the applications and also caters for their requests of insertion

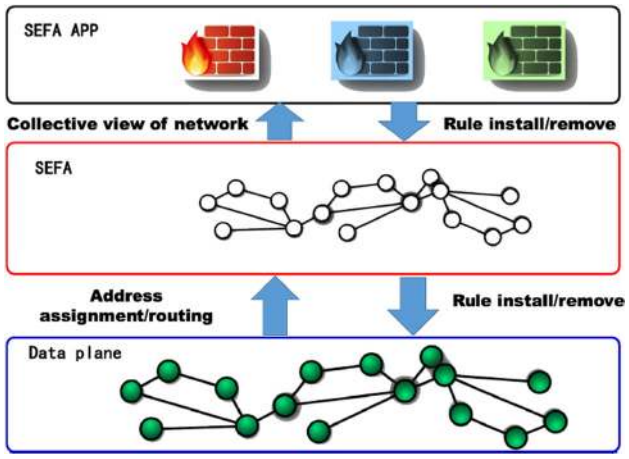


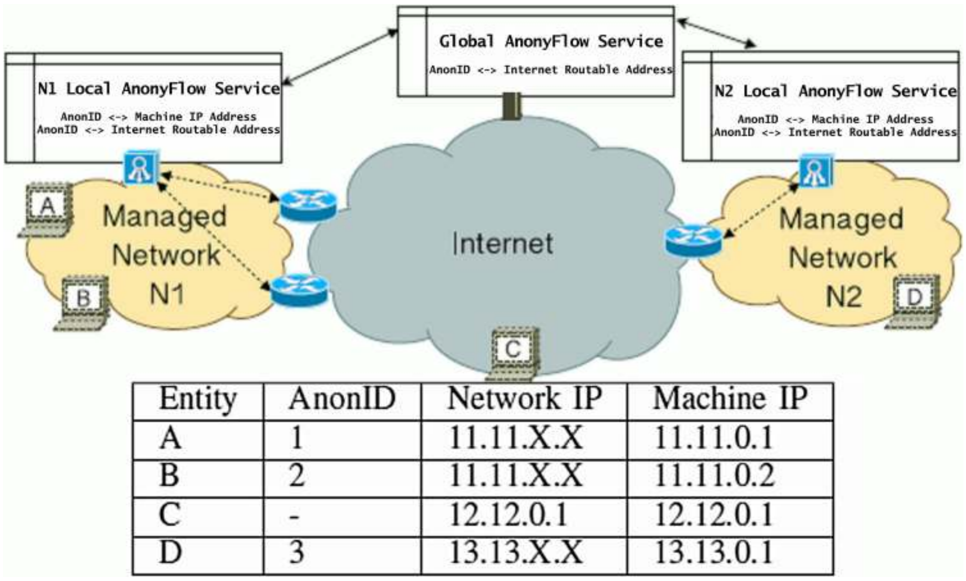
Fig. 12 Overview of SEFA architecture [17]

or removal of filtering rules. An overview of SEFA is given in Fig. 12.

2.2.2 Scanning and anomaly detection techniques

A solution to the defence against scanning and fingerprinting attack is presented in [18] as OpenFlow Random Host Mutation (OF-RHM). The dynamic and programmable nature of SDN is exploited to present a novel proactive Moving Target Defence (MTD) strategy. This strategy works by hiding the real IP addresses from the inside/outside scanning attacks. The key idea is to frequently mutate the host IP addresses and assign virtual IPs (vIPs) to the hosts with high rate of unpredictability and transparency from the end hosts. The real IP addresses are only accessible by authorised entities.

Suppose with the address ranges $r_1, r_2 \ldots, r_m$ and subnets $s_1 s_2 \ldots, s_z$, the problem of appropriate assignment of


Global AnonyFlow Service
AnonID <-> Internet Routable Address
N1 Local AnonyFlow Service
AnonID <-> Machine IP Address
AnonID <-> Internet Routable Address
N2 Local AnonyFlow Service
AnonID <-> Machine IP Address
AnonID <-> Internet Routable Address
A
Managed Network N1
B
Internet
C
Managed Network N2
D

Entity	AnonID	Network IP	Machine IP
A	1	11.11.X.X	11.11.0.1
B	2	11.11.X.X	11.11.0.2
C	-	12.12.0.1	12.12.0.1
D	3	13.13.X.X	13.13.0.1

Fig. 13 AnonyFlow operation [21]

ranges to subnet is considered as NP-hard [19] problem and is formulated using Satisfiability Modulo Theories (SMT) [20]. The problem of unpredictable and rapid mutation is established as a valid assignment of unused address ranges under multi-constraint satisfaction. Equation (2) dictates the mutation rate constraint and shows that total number of mutated *vIPs* of all hosts in subnet s_k during time T must be less than the aggregate size of all ranges assigned to s_k.

$$\forall\left(\sum_{1\leq i\leq n} c_{ik}R_i\right)*T\leq\sum_{1\leq j\leq m} b_{jk}\left|r_j\right| \tag{2}$$

Equation (3) defines the unpredictability constraint and establishes that the ranges must be assigned to subnets in proportion to their total required mutation rate.

$$\forall k,\ P_k=\frac{T*\sum_{1\leq i\leq n} c_{ik}R_i}{\sum_{1\leq j\leq m} b_{jk}\left|r_j\right|} \tag{3}$$

where P_k is the total required mutation of subnet s_k during time T on total range size allocated to it and P_a as given in Eq. (4) is the total required mutation of all hosts on total size of unused address ranges.

$$P_a=\frac{T*\sum_{1\leq i\leq n} R_i}{\sum_{1\leq j\leq m}\left|r_j\right|} \tag{4}$$

It has been shown through analysis and simulation that OF-RHM can invalidate the information gathering of external scanners up to 99 %. It can also save up to 90 % of network hosts from even zero-day unknown attacks

Mendonca et al. present AnonyFlow in [21] that provides anonymity to the users for the prevention of port scanning. The basic concept is to assign temporary IP addresses and flow IDs to the user traffic which is exiting their respective ISP's domain. The architecture of AnonyFlow consists of few identifiers notably *AnonID*, which is assigned to every node in the network for its communication with other nodes. It can either be used on per-flow basis or fixed for every service. Machine IP address and network IP address are two other identifiers used in this scheme.

This scheme is implemented as a service in OpenFlow-based networks using NOX controller. The AnonyFlow service consists of *AnonyFlow conduit, Local AnonyFlow service and Global AnonyFlow service.* AnonyFlow conduit is used to rewrite IP addresses to/from *AnonIDs*, and forwards the packets towards required destination. *Local AnonyFlow service* handles mappings within the network, and communicates with global service. *Global AnonyFlow service* is used for network lookup for *AnonIDs* outside local managed network.

AnonyFlow's operation is illustrated in Fig. 13 which elaborates the process when host 'A' opens a connection to a service on host 'B'. For this, 'A' sends a packet with source IP address '11.11.0.1' and destination '2' that passes through the AnonyFlow conduit on network 'N1'. The conduit consults with the *local AnonyFlow service* of 'N1' for the leading packet of the flow. The *local AnonyFlow service* determines that both the source address and the destination *AnonID* '2' is within the same network. It then rewrites the source address to '1' and destination to '11.11.0.2' and forwards the flow to the destination. If the destination *AnonID* is of another network as in case when host 'A' communicates with host 'D', the *local AnonyFlow service* must opt for a global lookup of the destination *AnonID* to determine a routable address to the destination network. The source address is rewritten to *AnonID* '1' when the flow arrives at a conduit in the desti-

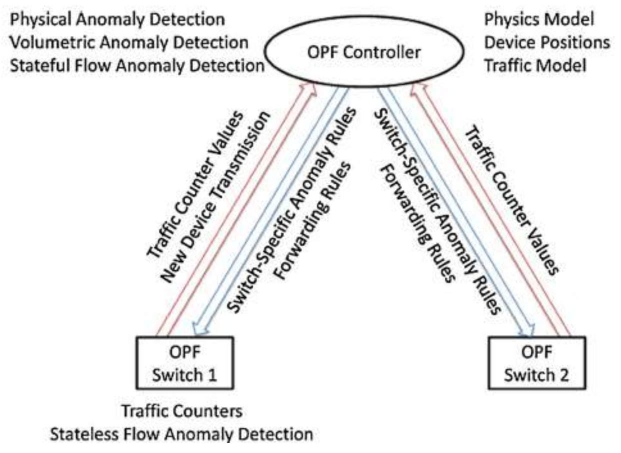


Fig. 14 Learning IDS controller/switch interaction [22]

nation network and the destination address is changed to the machine address of 'D'. The AnonyFlow service is compared to Tor-based anonymity service and result shows tremendous increase of throughput using AnonyFlow.

A learning intrusion detection system (L-IDS) based on SDN for protection of embedded mobile devices is presented in [22]. The L-IDS can be deployed within the existing network alongside existing security systems and suggests no modification to mobile devices. The proposed solution handles mobility of end-user devices and offers a wide variety of action space for responding to anomalies (flow dropping, re-authentication, honey pot redirection, and system isolation), which, however, traditional IDS solutions do not offer.

Implementation of L-IDS is based on a biomedical site in which implanted biomedical devices and robotic material transport are considered as a case study. These mobile embedded devices connect to the fixed infrastructure via the OpenFlow-enabled wireless switches deployed at specific points in the site. OpenFlow controller and the switch collectively work to detect different types of anomalies including *stateless, stateful, volumetric and physical anomalies* at different point in the network. Figure 14 shows the interaction of OpenFlow controller and switches, highlighting their contributions in terms of anomaly detection and the values exchanged for that purpose.

Another solution focusing on anomaly detection including TCP port scan detection, TCP and UDP flooding (DoS) attack detection is discussed in [23]. The authors have implemented a few well- known anomaly detection algorithms on SDN. Their basic idea is to place the Anomaly Detection System (ADS) in home networks built upon SDN instead of core network due to two problems at core (1) low detection rate versus large false positives, (2) inability to run at line speed.

Four well-known algorithms used for anomaly detection, namely Threshold Random Walk with Credit-Based Rate Limiting (TRW-CB) [24], Rate Limiting [25,26] Maximum Entropy Detector [27] and NETAD [28] are ported to judge their feasibility for SDN infrastructure. The experiments show that these algorithms are implemented in SDN and they process only a small fraction of the total traffic. However, they attain the equivalent accuracy achievable by inspecting every packet in conventional networks.

The TRW-CB algorithm detects scanning worm infections on a host by maintaining that the probability of a successful connection attempt should be much higher for a normal host than a malicious one. The algorithm maintains a list of new connection requests that have pending connection responses and increases the likelihood or probability of host being infected once the connection times out without a response. The likelihood is decreased if a successful reply is received. The implementation of TRW-CB in NOX leverages the basic concept of SDN which enables the new connection request to be forwarded to controller before the flow is installed in the SDN switches. Once a connection is successfully established and flows installed in the switches, the remaining packets are not further monitored. This enables the algorithm to achieve the accuracy comparable to inspection of every packet.

Rate Limiting uses the observation that an infected machine mostly attempts to connect with many different machines in a short span of time. An uninfected machine, on the other hand, attempts to connect at lower rate and mostly that too with recently accessed machines called 'working set'. If the connection is requested to a machine not present in the working set, it is put into a delay queue. The requests in the delay queue are taken out one by one every d seconds and forwarded to destination. Once connection successful message (i.e. *TCPSYNACK*) is received, the flows are installed to handle the rest of the traffic.

The Maximum Entropy detector estimates the benign traffic distribution using maximum entropy estimation. Traffic is divided into 2348 packet classes and maximum entropy estimation is then used to develop a baseline benign distribution for each class. The comparison of packet class distribution with the baseline distribution is made using KL (Kullback–Leibler) divergence measure [29]. If the KL divergence exceeds a threshold η_k, more than h time in last W windows of traffic, it raises an alarm for suspicious traffic.

The *NETAD* protocol works on a subset of traffic filtered as 'Uninteresting Traffic' based on the reason that a first few packets of a connection request are sufficient for anomaly detection. All incoming traffic, non-IP packets and TCP packets with sequence number more than 100, can be considered as a type of 'Uninteresting traffic'. To implement the same, for example, all packets having sequence number less than 100 are passed through the controller. Once the sequence number exceeds 100, the flows are installed in the switches and the traffic is passed without controller intervention.

The above algorithms are implemented in SDN environment and the efficacy of these in case of HOME/SOHO/ISP networks is evaluated based on NOX controller. The comparison among the different points of placement reveals that the detection is far more accurate in the home networks than the ISP.

2.2.3 DoS/DDoS detection and mitigation techniques

A lightweight DDoS flooding attack detection is proposed in [30] that uses Self-Organizing Maps (SOM) [31]. The problem is discussed as an unsupervised Artificial Neural Network (ANN) trained with features of the traffic flow. The algorithm frequently takes statistics about flows from the NOX registered switches as parameters for the SOM computations, which are then used to classify network traffic flows as either normal or sign of a potential attack. The proposed solution works in three modules as described in Fig. 15.

1. *The Flow Collector module* is responsible for periodically polling all Flow Tables of OF switches for flow entries over a secure channel, which is isolated from hosts connected to the switches.
2. *The Feature Extractor module* receives collected flows, extracts features that are important to DDoS flooding attack detection. It then gathers them in 6-tuples associated with a switch ID and passes it to the classifier.
3. *The Classifier module* uses SOM to analyse and classify whether a given 6-tuple corresponds to a DDoS flooding attack or to legitimate traffic.

The experimental setup is emulation based and the results are based on three parameters namely the (1) time to train and classify traffic, (2) detection performance and (3) Method

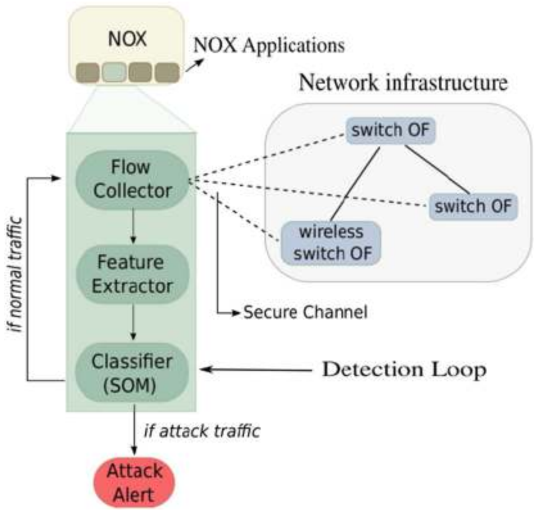


Fig. 15 DDoS detection loop using SOM [30]

overhead. The performance is compared to other approaches that are based on the KDD-99 [32] dataset and established that overhead in these approaches is due to the reason that they collect every packet sent to a victim, and then process this information to generate connection records. However, there is no discussion about the mitigation of the subject threats.

The authors in [33] have proposed a solution for DDoS mitigation that monitors OpenFlow statistics regarding traffic flows to detect indications of a DDoS attack. The first phase of DDoS mitigation is the initial detection, followed by the identification mechanism and then the blocking of the source of attack. The approach is fairly simple and starts with collecting flow statistics from the OF switches regarding the byte and packet counters polled every second. The difference between the current and previous value is added to data set Q with a limit of 60 entries. The standard deviation of this list (σ(Q)) is given in Eq. (5) as

$$\sigma(Q) = \sqrt{\frac{1}{60}\sum_{i=1}^{60}(Q_i - \mu)^2} \tag{5}$$

The anomalies that may potentially be signs of DDoS attacks can be detected by comparing the expected deviation and the real deviation in data set Q. Any sign of anomaly such as a sudden burst of data can trigger the identification mechanism to establish it as benign or DDoS attack. The identification of DDoS attack can be one of the two methods, (1) by sampling the specific flow, i.e. analysing the packet symmetry or (2) by temporarily blocking outgoing traffic and see which nodes continue to send traffic. Both methods are useful in identifying the source of DDoS attack with the later much useful in determining the spoofed sources.

The next phase is the blocking of all traffic from the identified sources of attack. The authors have performed experiments for their approach but have not compared their approach to any of the existing ones in literature. Therefore, it is difficult to conclude whether their solution is effective in detection and mitigation of DDoS attack and to what extent.

2.2.4 Forensic techniques

SDN's novel capabilities can be used to aid in better forensic analysis. The authors in [34] have introduced a Provenance Verification Point (PVP) component and have demonstrated the ability to detect the presence of attacks that were previously unobservable by conventional forensic systems.

The scenario elaborated in Fig. 16 is considered by the authors in which an administrator (Alice), running a data centre has observed unexpected behaviour within the network and now it is required to investigate whether the behaviour is legitimate or is indicative of a fault or attack. However, it

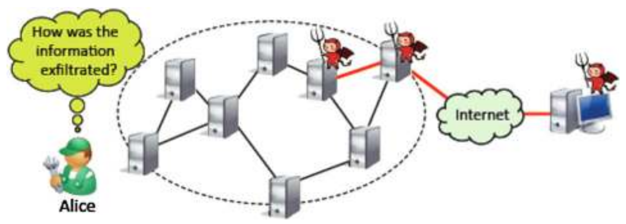


Fig. 16 Typical scenario of data loss from a network [34]

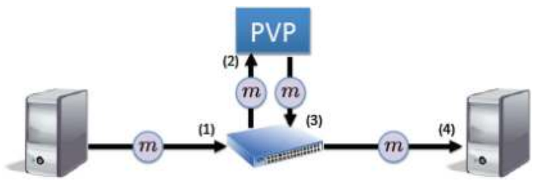


Fig. 17 PVP interaction as traffic interposition [34]

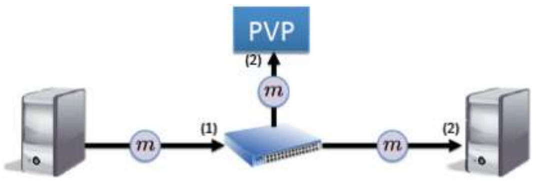


Fig. 18 PVP interaction as traffic mirroring [34]

is not feasible for Alice to simply ask the nodes about their activities as an attacker may have gained complete control of nodes and could return false information in response to Alice's request. As nodes in the network cannot be inherently trusted, a number of correct nodes are required. However, even if there are enough correct nodes available, they are often poorly positioned to observe the network interior and collectively they are unable to achieve complete observability of the network.

This problem of global observability is resolved using a set of middleboxes called Provenance Verification Point (PVP) collaborating with the SDN switches. The traffic is passed through these PVPs which monitor and report the statistics to the central controller. Each PVP is responsible for monitoring activity for some subset of system nodes. The architecture relies on forwarding traffic to middleboxes, and dropping traffic that does not adhere to a specified behaviour. The two basic forms of middlebox routing are the traffic interception and traffic mirroring and can be visualized in Figs. 17 and 18, respectively.

To achieve line speed processing of traffic at PVPs and to collect all necessary information at these points, PVPs are made to operate as a verification layer in an inter-node commitment protocol similar to PeerReview [35]. All inter-node communication, even between two faulty nodes, is monitored by the corresponding PVP, and an evidence for the communication is retained. Such capability allows PVPs to achieve full observability and catch a wider class of misbehaviour.

For example, Alice suspects a potential attack on her network around the time m was sent. She queries the nodes A and B in the network about m, its derivation, and its consequences. Consider the case in which A and B are both faulty and wish to hide m's presence. The PVP monitoring A and B possesses proof of m's transmission by A and receipt by B. Alice can demand a record of network activity from the relevant parts of A's and B's respective local logs, then compare the record to the PVP's list of authenticators. Alice will be able to detect the absence of the message and its acknowledgment, and declare both A and B to be faulty.

This SDN-based forensic tool offers many other features such as detection of covert communication, detection of inconsistence false claims by a node and automation of forensic analysis by introducing an upper layer script parsing.

2.2.5 SDN-enabled security framework

FRESCO [36] is an OpenFlow security application development framework and one of its kinds to facilitate the rapid design of OF-enabled detection and mitigation modules and their composition. FRESCO has addressed several key issues that when resolved can help accelerate the composition of new SDN-enabled security services. FRESCO has a built-in library of 16 basic reusable modules, and more sophisticated security modules can be built by connecting these basic modules. Each FRESCO module includes five interfaces, namely *input, output, event, parameter and action.* A wide range of essential security functions, such as firewalls, scan detectors, attack deflectors, or IDS detection logic can be fabricated using the basic modules and its properties.

FRESCO framework mainly consists of an application layer and an Security Enforcement Kernel (SEK) integrated into NOX controller. An overview of FRESCO architecture can be seen in Fig. 19. The application layer provides an interpreter and APIs to support application development and SEK is used to enforce the policy actions from developed security applications. The basic components of application layer are Development Environment (DE), Resource Controller and the FRESCO scripting language. The development environment offers services such as script-to-module translation, database management and event handling.

FRESCO's script language is used to instantiate and to define the interactions between different NOX security modules. These scripts invoke FRESCO's internal modules, which are instantiated to form any security application. The application is driven by the input specified via the FRESCO scripts and accessed via FRESCO's DE database API. These modules once instantiated are executed by FRESCO DE whenever the input events are received.

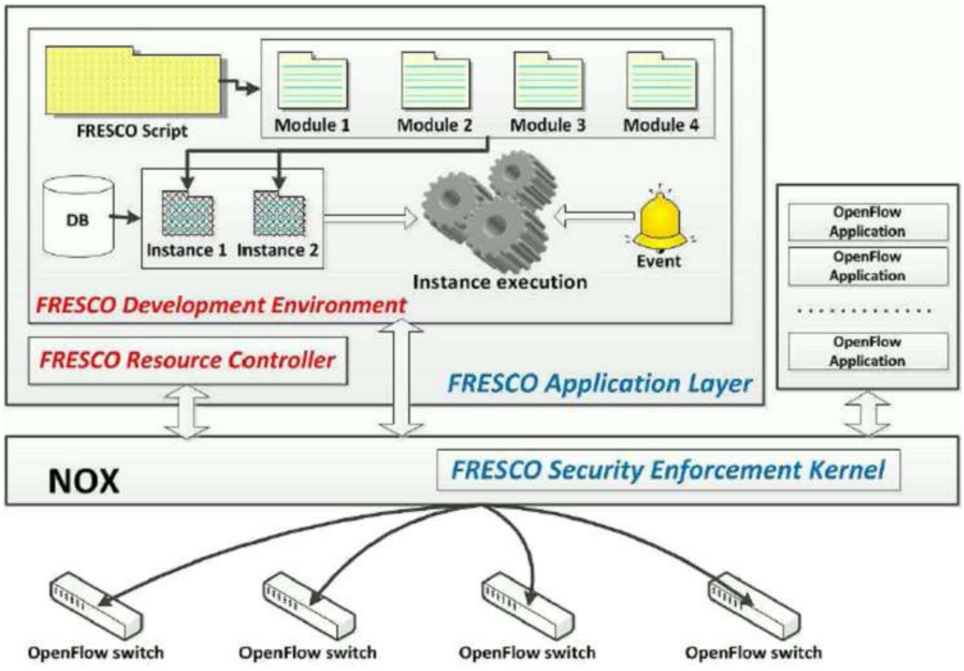


Fig. 19 Overview of FRESCO architecture [36]

Two case studies are presented to show the power and range of FRESCO. One application is used to detect and entrap malicious scanners and is named Reflector Net, which consists of a ScanDetector module and an Action Handler module. Scan Detector is triggered once it detects a large number of failed TCP connections and in turn invokes the ActionHandler to redirect malicious traffic towards a honeypot. Hence, the attacker continues to receive valid responses from the honeypot machine, under the impression that the attack is successful. In the second example, a use case of FRESCO's integration with a legacy security application is demonstrated. A BotHunter [37] application is used to detect a threat which in turn invokes the security applications written in FRESCO script to quarantine infected hosts on the network.

3 Survey of techniques for SDN protection

The advent of SDN has no doubt enabled a new vision of networking with its logical centralization of control and programmability, but has also opened up new avenues of threats due to the same. The security of SDN itself is still in infancy with not much work done in protecting the SDN and its components. This section highlights the contributions done so far in literature for securing the SDN.

3.1 Security analysis of SDN

A comprehensive study that highlights the major areas of concern for security of SDN is presented in [38]. The need for new responses to the imminent threats to the network

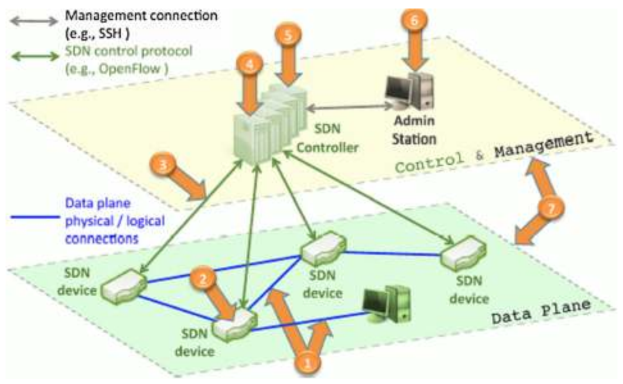

Management connection (e.g., SSH)
SDN control protocol (e.g., OpenFlow)
SDN Controller
Admin Station
Control & Management
Data plane physical / logical connections
SDN device
SDN device
SDN device
SDN device
Data Plane
1
2
3
4
5
6
7

Fig. 20 SDN threat vector [38]

is argued by giving a generalized description of threats to seven key areas of SDN as shown in Fig. 20. The threat vector associated with these seven areas are (1) Forged or fake traffic flows, (2) Attacks on vulnerabilities in switches, (3) Attack on Control Plane communication, (4) Attacks on/vulnerabilities in Controllers, (5) Trust between the controller and the applications, (6) Vulnerabilities in Admin station and (7) Trusted resources for forensics and remediation.

A security analysis of SDN based on a hybrid combination of STRIDE [39] and Attack trees [40,41] is given in [42]. The work is based on the security analysis of OPENFLOW protocol and is confined to the data plane only. A secure communication is assumed between the controller platform and the SDN switches. As the approach is confined to the data plane, the only focus is on addressing the threats that can exploit the vulnerabilities related to data flows within the SDNs. At first, the focus is on the denial of service attack against the flow table, arguing and evaluating that the flow table can be flooded by flow rules. With respect to Information Disclosure, the attacker may be able to derive information about network state, e.g. active flow rules, by observing differences in controller response times. At last, the Tampering attack is discussed and the possibility of cache poisoning attacks against the flow table and/or controller state is suggested.

The security analysis and practical evaluation is followed by recommendations for mitigation of subject threats. The authors suggest that DoS attack, though possibly hard to detect, can be mitigated by rate limiting, event filtration, packet dropping and timeout adjustment. At the same time, access control and flow aggregation might help limit the chances of DoS attack. The outcomes of timing attack in the form of information disclosure can be dealt with proactive flow rule establishment and by increasing the variance of measureable response time for flow rule insertion. However, there are no suggestions regarding counter-acting tampering attacks.

3.2 Threats to SDN

Similar to the evolved nature of SDN, the threats in SDN framework have also evolved into more complicated phenomena that can take advantage of the SDN framework in many new ways. The authors in [43] have contributed a way to fingerprint SDN networks. Their idea is based on the strategy that the difference in response times for existing flow and a new flow in SDN network can be exploited by the attacker to gain knowledge about the network. A custom SDN scanner as shown in Fig. 21 is built that works on the same principle and collects a set of values for response times.

The statistical differences, i.e. mean and standard deviation values are sufficient to gain knowledge about the SDN network. Once it is established that the network in question is the SDN network, it is fairly simple to attack resource consumption or DoS attack against the control and data plane. The work is limited as of just identification of SDN network and needs further research in the area of fingerprinting and scanning of SDN networks to know about the topology of the target network. Moreover, they have not presented any solution to defend the SDN network from the fingerprinting attack.

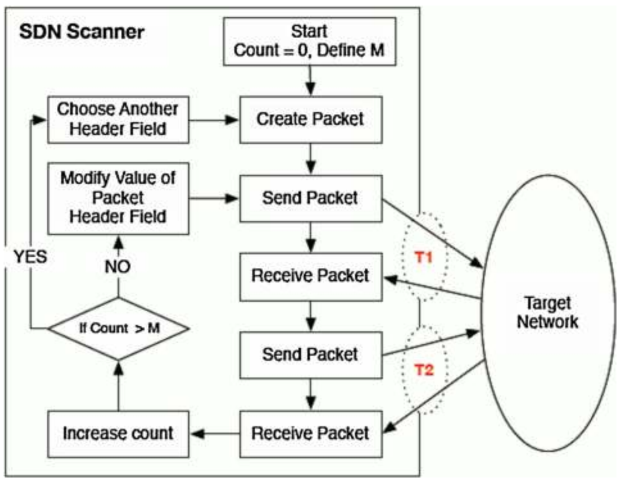


Fig. 21 Function diagram of SDN Scanner [43]

Another attempt to thwart the SDN topology is an ARP cache poisoning attack specific to SDN topology poisoning along with a solution to such attacks is presented in [44]. With the poisoned network visibility, the upper-layer OpenFlow controller services and applications are totally misled, leading to serious hijacking, denial of service or man-in-the-middle attacks. The vulnerabilities in the Device Tracking and Link Discovery Services of main stream SDN controllers are exploited to present two types of topology poisoning attacks, i.e. Host Location Hijacking Attack and Link Fabrication Attack. These attacks are further used to launch Man-In-The-Middle (MITM) and DoS attacks.

Two possible mitigation methods to secure Host Tracking Service (HTS) in SDN controllers are discussed with the ability to dynamically track network mobility. One possibility is to generally authenticate host using PKI infrastructure, and particularly whenever the host changes its location. The other solution is to explicitly verify the legitimacy of the migration before allowing services to host. Link fabrication attack is avoided by strengthening the vulnerabilities in Link Discovery Service. One possible way of doing this is to authenticate the Link Layer Discovery Protocol (LLDP) packets, and the other method is by verification of switch port property.

3.3 Network consistency solutions

FlowChecker [45] identifies misconfigurations within the OpenFlow networks. It uses Binary Decision Diagrams (BDD) to model the switch flow-table configuration in the form of state machine. It uses Computational Tree Logic (CTL) to write queries for verifying properties. FlowChecker then uses the model checker technique to validate correctness of the interconnected network by pair-wise comparison of each pair of flow table rules in the domain.

FlowChecker is defined as an independent centralized server application that receives queries from OpenFlow applications to verify, analyse or debug OpenFlow configuration. These queries could be limited within one domain or across different federated domains. FlowChecker can run on a master controller (spans number of federated OpenFlow infrastructures) and it uses a special protocol to communicate with other controllers and switches.

Figure 22 describes a possible scenario for validating federated domains. The numbers assigned in the diagram are

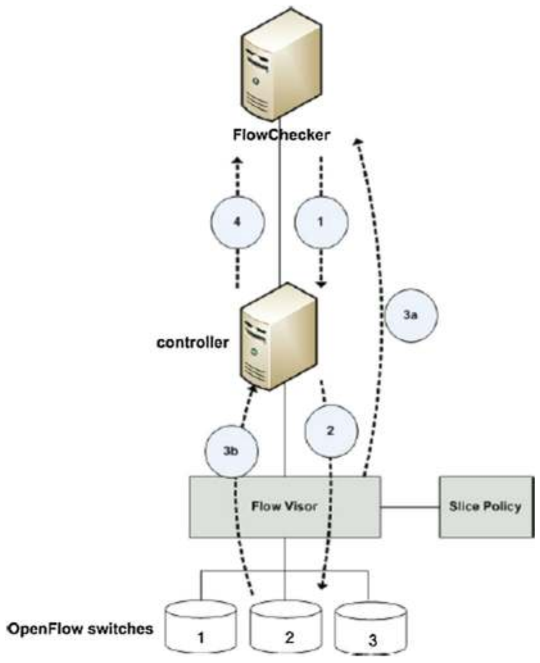


Fig. 22 FlowChecker interaction [45]

stepwise functions performed for validation. A user query written in CTL logic to check the validity of certain properties is received by the FlowChecker which sends a special message to the controller(s) asking for extracting FlowTables and sending them back to FlowChecker. When controller(s) receive the message, a request is sent to the respective OpenFlow switches to provide their FlowTables. The message sent from a controller to an OpenFlow switch is intercepted by the FlowVisor, which responds to this message directly to the FlowChecker by sending "slices policies" for that domain. FlowTables are sent to the FlowChecker once they are received by a controller. The FlowChecker has now got the suggested policies, "slices policies" and the "FlowTables". The suggested "slices polices" are compared with FlowTables for validation and a report is generated to indicate the conflicts and misconfigurations also to reply the CTL query.

FlowChecker can also be used to analyse the impact of applications on the network before installation. Experimental evaluation follows the description of FlowChecker in which randomly generated flow tables of various sizes are overlapped, and analysed for configuration conflicts. Results show that FlowChecker takes approximately 160 ms to verify correctness in a network of 120 switches and flow tables having 1000 rules.

A role-based authorization and security constraint enforcement solution for NOX OpenFlow controller named as FortNOX is presented in [46]. FortNOX solves a critical challenge of efficiently detecting and reconciling potentially conflicting flow rules imposed by dynamic OpenFlow (OF) applications. It enables NOX to check flow rule contradictions in real time. The algorithm is robust even in cases where an adversarial OF application attempts to strategically insert flow rules that would otherwise circumvent flow rules imposed by OF security applications.

The architecture consists of a security policy enforcement engine running a novel algorithm called *alias set rule reduction*. A role-based authentication is recommended for security authorization of each OF application using digital signatures. A privilege value is assigned to the flow rule of each application, whereas unsigned applications have the lowest priority. The second part of the algorithm is the conflict analyzer. It is used to analyse conflict for a newly installed flow rule against those in the aggregate flow table.

Another important feature of FortNOX is the Security Directive Translation that mediates a set of high-level threat mitigation directives into flow rule. These flow rules are then digitally signed before submitting to FortNOX. There are seven security directives, namely *block, deny, allow, redirect, quarantine, undo, constrain* and *info*. The most important of these seven are the *redirect*, *quarantine* and *constraint* directives. The *redirect* directive enables a security application to tunnel all flows between a source and given target to a new destination, e.g. diversion of malicious traffic into a honey pot. The quarantine directive enables the security application to isolate the host from the entire network thus preventing it to originate any flow. The constraint directive enables the security application to deactivate the current flow rules that are not set to a specific priority. This helps to rate-limit the flows in case of DDoS attack.

No bugs In Controller Execution (NICE) [47] uses automated model checking to discover errors in SDN applications. The details about the controller, network topology specification, number of switches and hosts are input to NICE for testing applications. These inputs automatically generate specific traffic flows to study the network for a variety of different events, and recognize property violations. The logical placement of NICE and its components in OpenFlow-based network can be seen from Fig. 23. NICE checks for user specified 'correctness properties' which include no black holes or direct paths, no forwarding loops, no forgotten packets, etc. A novel idea of utilizing symbolic execution is employed to avoid the rapid expansion in state space that usually arises when model checking is used. NICE models the impact of a subset of representative packets of all the different packet classes instead of exploring all possible states in the network.

NICE is implemented for the NOX controller using Python to test real applications and uncovers several bugs. Experimental results indicate that NICE is five times efficient than existing model checkers. This shows that finite-state modelling is a relatively simple verification technique which can be applied to various types of applications. However, it is challenging to scale this approach to large networks. In appli-

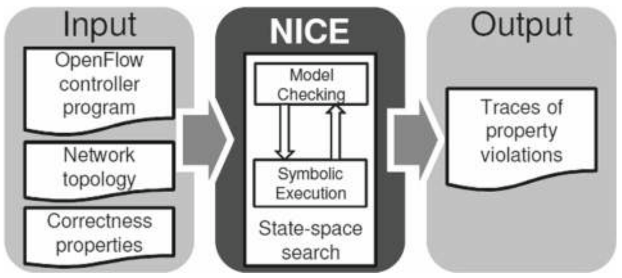


Fig. 23 Logical placement of NICE and its components [47]

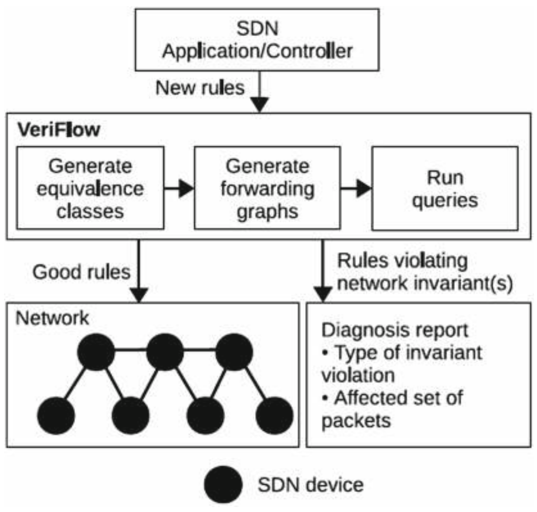

SDN Application/Controller
New rules
VeriFlow
Generate equivalence classes
Generate forwarding graphs
Run queries
Good rules
Rules violating network invariant(s)
Network
Diagnosis report
• Type of invariant violation
• Affected set of packets
SDN device

Fig. 24 VeriFlow architecture [48]

cations with infinite states, this approach cannot definitively prove the lack of errors.

Another novel approach for network consistency verification is VeriFlow [48]. It is a dynamic solution which verifies network correctness in real time. The key challenge is to keep the network latency to a minimum possible value when performing verifications as the network evolves. VeriFlow works as layer between the controller and the switches and checks the validity of network invariants whenever a new forwarding rule is installed.

VeriFlow's primary job is to keep track of every forwarding-state change event. This is done by dividing the network into a set of equivalence classes on the basis of existing rules. Packets belonging to a certain class undergo the same forwarding decisions throughout the network. When it is time to introduce a new rule, the classes that will be altered are located and network invariants are verified within those classes. VeriFlow maintains individual forwarding graphs for the equivalence classes and traverses them to query the invariants. Each flow modification is thus verified in real time before it is implemented. VeriFlow then traverses these graphs to determine the status of one or more invariants. The logical structure of VeriFlow is shown in Fig. 24.

The VeriFlow prototype implementation is run using a NOX controller managing a simulated OpenFlow network on Mininet [49]. It shows that VeriFlow has minimal impact on network performance and it takes only a few hundreds of microseconds to verify network-wide invariants. However, as per recommendation from the authors, it is not yet feasible to implement VeriFlow with multiple controllers as it is difficult to obtain a complete view of network state in that case.

3.4 Trust management solutions

A dynamic trust model is proposed in [50] arguing the requirements for software attestation and trust management. The authors put forward and exhibit the feasibility of a model to support autonomic trust management in component-based software systems. Their model allows a trustor to assess the trustworthiness of the trustee by observing its behaviour. Quality attributes such as Availability, Reliability, Integrity, Safety, Maintainability, and Confidentiality are considered as the parameter to measure the behaviour.

Another solution for trust management between OpenFlow application and controller is given in [50]. The authors propose PermOF, which isolates the controller kernel modules at runtime thus preventing the applications form calling on them directly. It also defines a set of fine-grained permission categories namely *Read, Notification, Write and System*, which allow sharply defined access privileges to applications. It is argued that enforcing security policies is a deterministic and low-cost solution for over-privilege problem in applications.

4 Security modelling

Threat or security modelling is a procedure for identifying system objectives, associating known or foreseen vulnerabilities and then defining countermeasures to prevent, mitigate or minimize the effects of threats to the system. There are three general approaches to threat modelling namely at Attacker-centric, software-centric and asset-centric.

4.1 Attacker-centric

Attacker-centric threat modelling starts with an attacker, and evaluates their goals, and how they might achieve them. Attacker's motivations are often considered which are broadly categorized into either stealing information or jeopardizing the system and the same is the reason this approach starts from either entry points or assets.

Modelling using attack tree [40] is one such example of the same in which any attack against a system is represented in a tree structure, with the goal as the root node and different ways of achieving that goal as leaf nodes. Complex attack

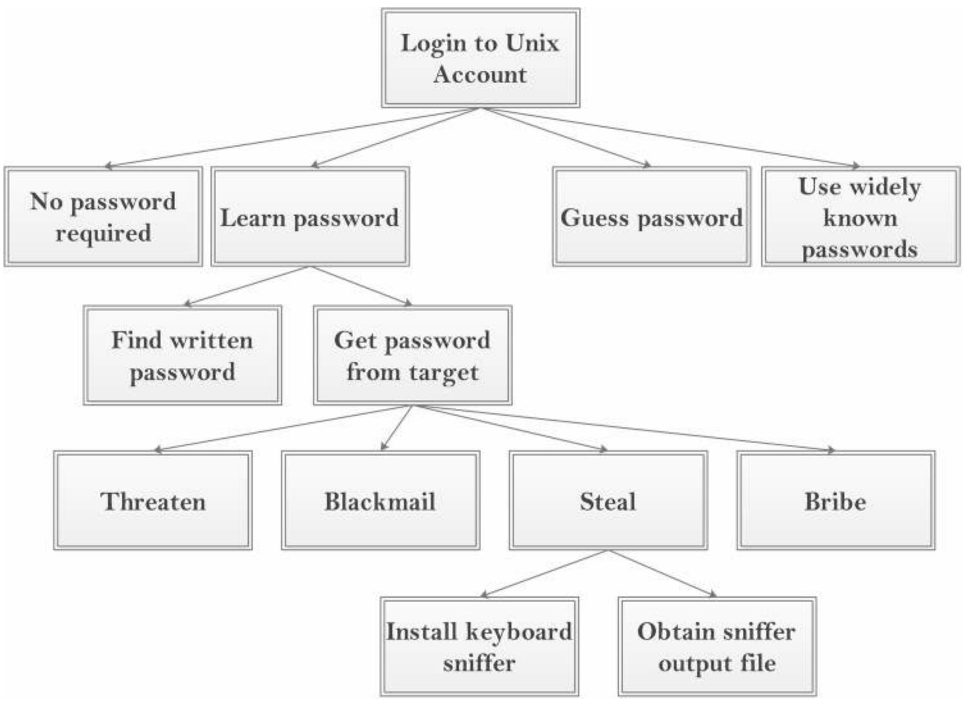


Fig. 25 Attack tree for login to a UNIX account [52]

trees with numerous sub-goals can further be divided into separate trees for convenience. There is an AND/OR logic between every node depending on how these nodes (attack method) contribute to their parent node (goal/sub-goal). Figure 25 represents an illustration of attack tree for login to a UNIX account [52].

4.2 Software-centric

Software-centric threat modelling (also called 'system-centric,' 'design-centric,' or 'architecture-centric') begins with the design of the system, and attempts to step through a model of the system, looking for types of attacks against each element of the model. Microsoft's STRIDE [39] model is one example of the software-centric threat model.

The basic goal of security in a system is to ensure Confidentiality, Integrity, Availability, Authentication, Authorization and Non-repudiation. One way to ensure that the application have these properties is to employ threat modelling using STRIDE, an acronym for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. It is a security model focusing primarily on the application security aspects of the system.

The system under observation is decomposed into relevant components and the inter-component interaction is highlighted using Data Flow Diagram, UML or any other representation. Each component is then analysed for susceptibility to above six threats. What follows is the mitigation of the threats by introducing security components. The process is then repeated until threats are alleviated and a comfortable argument can be passed for the system being secure.

Primarily, the STRIDE model is developed for application and software security but can be expanded to the system level. However, it is unjustifiable to use the same for networks as the entities in network are inherently different from that of applications. For example, to mitigate the threat of information disclosure between two software components, the most common way is to upgrade your APIs to the ones that support secure protocols such as SSL/TLS or IPSEC. However, the same upgrade is not available in network components at easy cost. The solution is either a device upgrade or installing a new hardware that comes with its own piece of vulnerabilities.

4.3 Asset-centric

Asset-centric threat modelling starts from assets entrusted to a system, such as a collection of sensitive personal information. Assets are classified according to data sensitivity level and their value to a potential attacker, to prioritize risk levels. Although obsolete but Microsoft's DREAD [50] model is one example of asset-centric security evaluation of the software system.

DREAD is used to quantify, compare and prioritize the amount of risk presented by each evaluated threat. The DREAD acronym is formed from the first letter of each category as Eq. (6)

$$\textbf{Risk\_DREAD} = (\underline{D}\text{AMAGE} + \underline{R}\text{EPRODUCIBILITY} + \underline{E}\text{XPLOITABILITY} + \underline{A}\text{FFECTED USERS} + \underline{D}\text{ISCOVERABILITY})/5. \quad (6)$$

The calculation always produces a number between 0 and 10. Higher numbers indicate more serious risk.

4.4 Hybrid models

Despite the usability and flexibility of both models presented by Microsoft, however, there exist a few less flexible or limited threat modelling techniques such as TRIKE [54], AS/NZS 4360:2004 Risk Management [55], CVSS [56] and OCTAVE [57].

5 Antifragile cyber security model

In this section, we propose a cyber security model for SDN framework as a unified approach to put together all elements of securing SDN- enabled systems. The concept of antifragile cyber security is introduced and the application of this model in this perspective is also outlined.

5.1 Fundamentals of cyber security model

The fundamental concept of our model suggests seven key threats to SDN-enabled networks while connecting to the cyber world. These are (1) Scanning/Enumeration/Probing, (2) Spoofing, (3) Sniffing, (4) DoS/DDoS flooding, (5) Unauthorized Access/Elevation of Privileges, (6) Malwares/Logic Bombs and (7) Anti-Forensic Techniques. Our proposed model merges the need of securing SDN itself with the security services offered by SDN into a single entity and is the first step towards achieving the overall objective of antifragile cyber security.

5.1.1 Scanning/enumeration/probing

The very basic of an attack starts with a scanning or probing attempts against the target. Here, the word scanning, probing and enumeration are used interchangeably in our proposed model. Probing can be done in many ways but the basic goal is always the same, i.e. to know the maximum about the network and its services.

We propose that the following different kinds of probing attack can affect the SDN network as shown in Fig. 26,

1. Network scanning—used to visualize the network, such as placement of controller, switch architecture and management stations.
2. Port scanning—used to gain knowledge about the services running within the network, applications using specific ports can be revealed through port scanning. Target systems are SDN controller and application servers.
3. Vulnerability scanning—application servers and SDN controller are potential victims. Host operating system's vulnerabilities serve as primary target in such scanning attack.

Possible countermeasures include different ways to hide the identity of hosts in the network, host hardening and patching for possible vulnerabilities

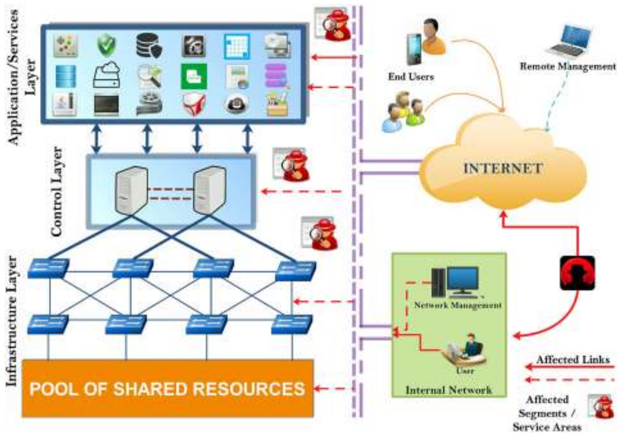


Fig. 26 Affected segments from Probing Attack

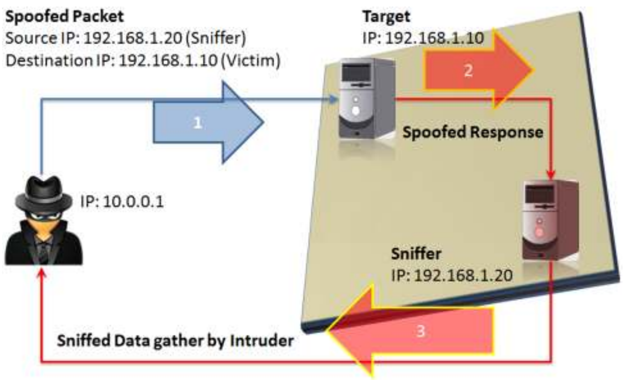

Spoofed Packet
Source IP: 192.168.1.20 (Sniffer)
Destination IP: 192.168.1.10 (Victim)
Target
IP: 192.168.1.10
1
2
Spoofed Response
IP: 10.0.0.1
Sniffer
IP: 192.168.1.20
3
Sniffed Data gather by Intruder

Fig. 27 A typical IP spoofing attack

5.1.2 Spoofing attack

A malicious party may impersonate another device or user on a network to launch attacks against network hosts such as Man-in-middle and Denial of Service attacks, steal data, spread malware or circumvent access controls. Some of the most common methods include IP/Port spoofing shown in Fig. 27, ARP spoofing and MAC (device) spoofing attacks as shown in Fig. 28.

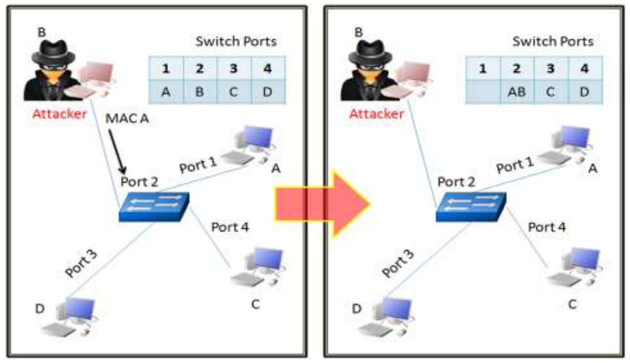


Fig. 28 A basic MAC spoofing/ARP cache poisoning attack

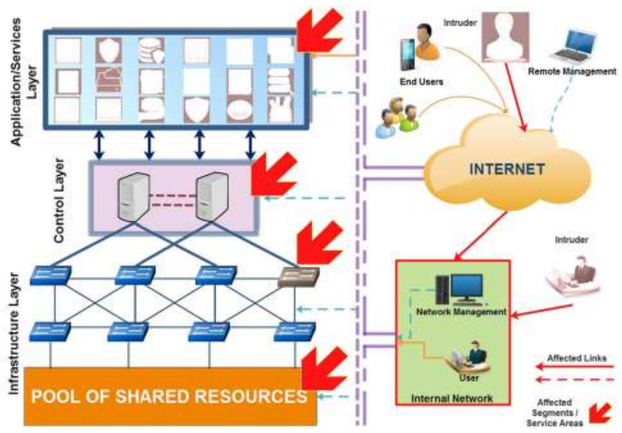


Fig. 29 Possible areas affected by spoofing attack (attacker impersonates as a legitimate user)

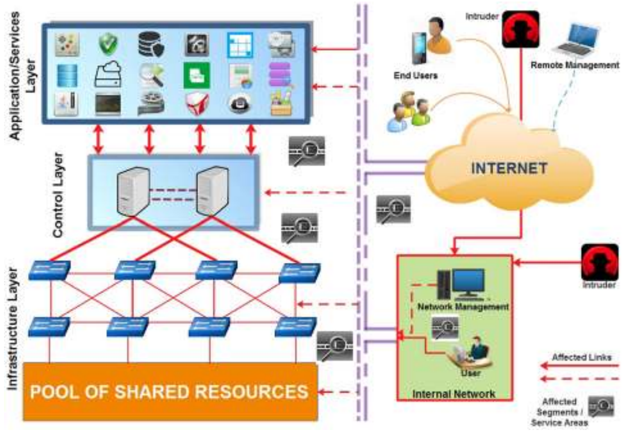


Fig. 30 Possible areas of sniffing attack

We envision that SDNs are no exception to these spoofing attacks and when it comes to securing them, there is still a lot of work required to be done.

Our model proposes the following types of possible spoofing attacks on the SDN-enabled networks, and is illustrated in Fig. 29.

1. *IP/Port Spoofing:* Can lead to DoS/DDoS attack against the infrastructure including the controller.
2. *ARP Spoofing/Poisoning attack:* A malicious party can use ARP spoofing to steal information, modify data in transit or stop traffic on a network. The network is poisoned with false ARP information so that the legitimate communicating parties unknowingly start transmitting data to the adversary. We envision that a successful attack on SDN can effectively poison the network topology information. With the poisoned network visibility, the upper-layer OpenFlow controller services and applications may be totally misled, leading to serious hijacking, denial of service or man-in-the-middle attacks.
3. *MAC Spoofing:* Impersonating a device using the legitimate device identity, e.g. MAC address can seriously downgrade the network performance and may lead to data stealth as well. We suggest that a rogue switch or a rogue machine may mislead the controller to make incorrect decisions which may jeopardize the performance of overall network. An impersonated SDN controller can wreak havoc and can lead to more serious consequences.

5.1.3 Sniffing

Sniffing involves capturing, decoding, inspecting and interpreting the information inside a network packet on a TCP/IP network. The purpose is to steal information, usually user IDs, passwords and network details. Sniffing is generally referred to as a "passive" type of attack, wherein the attackers sit silent/invisible on the network which makes it difficult to detect, and hence it is a dangerous type of attack. Figure 30 shows the proposed areas of SDN affected by sniffing attack. Sniffing can range from Layer 1 through Layer 7 depending on the type of information required by the attacker. It is very hard to detect a sniffer on network because its activities are quiet and powerful and many times there are no traces left to determine the presence of a sniffer.

There is no effective solution which can be used to defend against the sniffer's installation and attack on a network. Limiting broadcast information and using secure transfer protocols may limit the dangers of the sniffing attacks. However, the prelude to sniffing is the spoofing attack and hence, in our recommendation if avoided, may lead to avoid the sniffing attack.

5.1.4 DoS/DDoS flooding

DoS/DDoS attack is considered a nightmare for networked systems. Leveraging the SDN's capabilities of providing security as a service shows great promise for mitigation of DoS/DDoS attacks. However, SDN's own infrastructure is

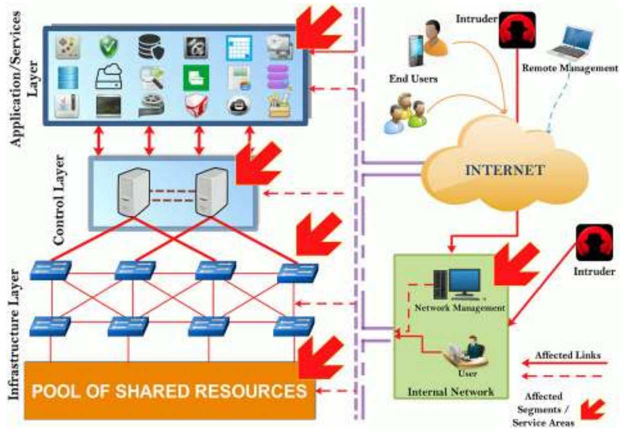


Fig. 31 Possible avenues of DoS/DDoS attack

highly susceptible to DoS/DDoS attack at every layer of abstraction, i.e. Application, Control and Infrastructure layers, but very little work has been done in this regard. We suggest that there is a need of an integrated solution that is able to cope with this dilemma of SDN being both the defender and victim of DoS/DDoS attacks.

Figure 31 shows the possible areas affected due to a DoS/DDoS attack. There are numerous DDoS attack methods used to degrade the overall performance of the services associated with the target network. These methods can be classified as

1. Volume-based attacks such as UDP floods, ICMP floods, and other spoofed-packet floods that can target the network links in SDN.
2. Protocol level attack such as SYN floods, fragmented packet attacks, Ping of Death, Smurf DDoS can target the switches and controller in SDN.
3. Application layer attacks include Buffer overflow and Zero-day attack and can affect the SDN-enabled applications.

5.1.5 Unauthorized access/elevation of privileges

Unauthorized access, access in excess of authorization and elevation of privileges are the class of attack through which a bug, design flaw, or a misconfiguration in software or the operating system is exploited to gain privileged access to the system resources.

We suggest that programmability of SDN by applications leaves room for misbehaving applications fiddling with the underlying network. Vulnerabilities in application may help an adversary to not only compromise application but the entire network. A compromised or malicious application running on top of SDN can access system or network resources beyond its access. For example, a poorly configured web application running on a specific operating system may allow the entire OS and other running application sharing the same server to be compromised. Figure 32 shows the possible points where a rogue user can escalate privileges to gain unauthorized access to system resources not meant for it.

5.1.6 Malwares/logic bomb

Although malwares have remained a consistent threat throughout the history of software, their functioning and potential to damage the victim has changed ever since. Benign applications may contain pieces of code planned to execute at some fixed point in time, i.e. logic or time bomb.

Code or software containing a logic bomb may not be detected by traditional antimalware tools. These tools use custom code designed for a particular system and scenario and hence no signature for logic bomb exists to detect them. We believe that in any data centre network, with application-driven network infrastructure, it would be nightmare to see an application suddenly behaving maliciously and wreaking

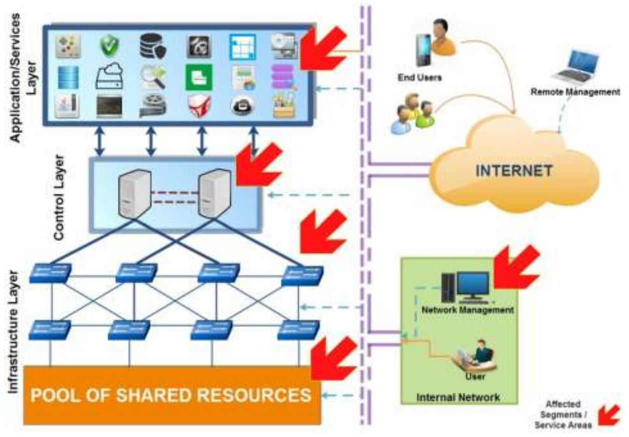


Fig. 32 Possible points of privilege escalation

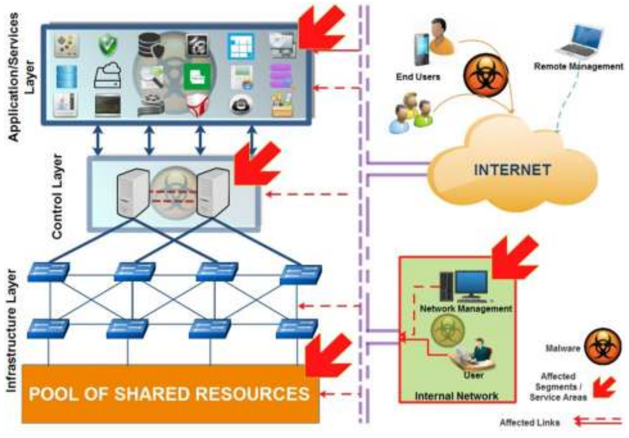


Fig. 33 Possible avenues of attack by Malwares

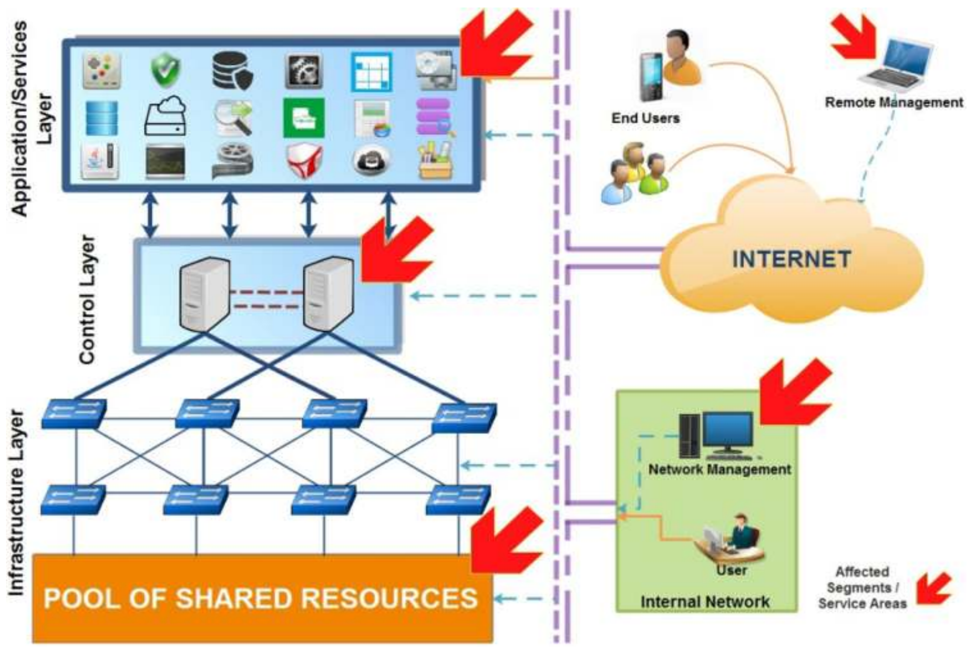


Fig. 34 Possible threat avenues of anti-forensic techniques

havoc in the network with the first step as confusing the controller.

Antivirus softwares and IDPS can be a solution to detect and prevent damage from malwares. However, we suggest that the need of hour is to think about application trust level in addition to other mechanisms. An application communicating with controller to gain access to the SDN network resources must be first signed and trusted. Figure 33 illustrates the possible victim areas when a malware spreads across the network through infected nodes.

5.1.7 Anti-forensic techniques

Anti-forensic techniques range from as little as removing or manipulating the traces of attacks to something more strenuous as attacking the forensic tools. One of the most widely accepted subcategory breakdowns was developed by author in [66]. He has proposed the following sub-categories, (1) data hiding, (2) artefact wiping, (3) trail obfuscation and (4) attacks against the computer forensics (CF) processes and tools.

To investigate and establish facts about an incident, we suggest that reliable information from all components and domains of the network is required. Furthermore, this data will only be useful if its trustworthiness (integrity, authenticity, etc.) can be guaranteed. Similarly, remediation requires reliable system snapshots to bring network elements to their working state. The potential areas affected when anti-forensic techniques are in place to risk the availability of logged data can be seen in Fig. 34.

5.2 Antifragile cyber security

Our proposed cyber security model has suggested the overwhelming threats and potential victim areas within SDN-enabled system and is summarized as Table 2. Applying this proposed cyber security model can be a repetitive process evolving with the passage of time as new vulnerabilities become known in systems and new threats start hovering over the horizon of information. A classical cycle of security management Plan, Protect and Respond (PPR) [58] is shown in Fig. 35. This cycle starts from the very principal of planning with identifying vulnerabilities, threats and associated risks to information. These risks are then covered by deterring the related threats by possible means and standing by for a possible response in case of an incident. This method of protection continues till the lifecycle of the system and after every response there happens to be always another phase of planning, protecting and responding.

Within the PPR cycle, there are methods and procedures involved along with a number of underlying tools and techniques. They have characteristics that are employed to prevent, detect, deny and respond to threats imminent to our information. However, apart from the discussion on their usage, they always try to facilitate the above-mentioned PPR cycle to work toward a single goal of realizing a system that is resilient in case of cyber threats.

The advent of modern technologies such as Internet of Thing (IoT) and wireless sensor networks (WSNs) deems it necessary to build systems that are less prone to errors. A trend of reducing human intervention, in case of recov-

Table 2 Summary of antifragile cyber security model for SDN

Sr.	Threats	Sub-categories	SDN specific
1	Scanning/Enumeration/Probing	Network scanning Port scan Vulnerability scanning	Different types of flows can be originated within SDN to know the placement of SDN controller(s) and to know about the topology
2	Spoofing	IP-Port spoofing	Fake or forged flows, Pre-Req to DoS
		ARP spoofing	Cache poisoning, MiTM
		DNS spoofing	Rogue applications
		MAC (device spoofing)	Rogue switch, rogue controller
3	Sniffing	Network	(1) Application to controller,
		Protocol	(2) Controller to switch and
		Packet	(3) Switch to host communication
4	DoS/DDoS floods	Volume-based attacks	Saturates bandwidth
		Protocol attacks	Saturates hardware or server resources
		Application layer attack	Buffer overflow, Zero day attack etc
5	Unauthorized Access/Elevation of Privileges		(1) Programmability of SDN by applications leaves room for misbehaving applications that may try to access system or network resources beyond its access (2) Vulnerabilities in application may help an adversary to not only compromise application but the entire network
6	Malwares/Logic Bomb	Virus	Can corrupt or remove files, spread to other systems and attaches itself into files and other programs
		Worms	Can replicates functional copies by exploiting vulnerabilities in targeted systems
		Trojans	Can be hidden inside applications that appear to be safe
7	Anti-Forensic Techniques		SDN provides a much reliable view of activities within a network. It can be used as a point of observation for investigating faults within a network

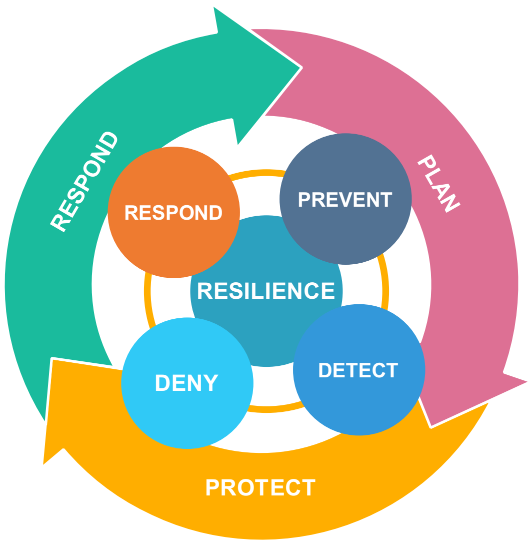


Fig. 35 Classical PPR cycle

ering from faults, errors and other damages, is seen for these systems. The literature presented in [59–63] are a few cases advocating the need to induce knowledge and intelligence in modern systems. These intelligent systems, when deployed with minimum knowledge, must learn from their environment in the wake of challenges and obstacles, making mistakes and growing stronger as time passes by. This concept of growing stronger is the exact depiction of antifragility and such intelligence can be added in our security-related systems, techniques and tools to let them learn from their past behaviours.

Keeping in view these facts, we propose to apply this concept to the domain of cyber security in the same way as that of any other realm so as to move beyond resilient security towards antifragile cyber security.

The concept of proposed antifragile cyber security can be seen in Fig. 36. The security mechanisms, i.e. tools, techniques and methods, underlying the conventional PPR cycle

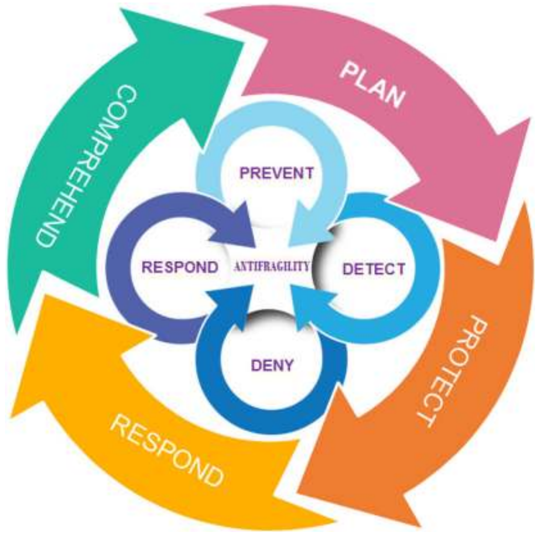


Fig. 36 Antifragile cyber security

are modified so that they are now iterative in nature. They can now gain from their knowledge base to grow with the passage of time. These techniques have the behaviours necessary to be independently antifragile in addition to facilitate and correspond with other techniques and methods. An example of this solution can be thought of as a technique for detection of DDoS attack with a behaviour and pattern recognition system that continually evolves with passage of time. This DDoS detection technique can be used to complement prevention and mitigation of DDoS attacks.

The conventional PPR cycle may not suffice for our proposed elastic behaviour of security mechanisms. If the underlying techniques are meant to grow but not the process itself, this might not be a complete antifragility. We suggest adding new methods for learning and comprehension to this cycle, as shown in Fig. 36. This way we can accommodate the cognitive behaviour of underlying security mechanisms to achieve the overall goal of a complete antifragile cyber security mechanism. SDN happens to be just the right way to realize it.

In conventional networks, there are disintegrated points of knowledge, due to different hardware, software and configurations. The problem of collecting and correlating networking data, specifically related to security, increases exponentially with the network expansion. Contrary to that, SDN's central control of network knowledge has the capability to learn and offer learning to other processes to aid them develop cognitive behaviour. Although very little work has been done with this perspective in mind, but our proposed Antifragile Cyber Security Model can be founded on the principal of security provisions offered by SDN. However, it eventually lacks the strength in the absence of securing SDN itself. The techniques mentioned in Sect. 2 are summarized as Table 3 in terms of their contribution towards antifragile cyber security.

Table 3 Security enhancement with SDN offering antifragility

Summary of security enhancement using SDN

Category	Technique	Antifragility
Monitoring systems and middleboxes	SLICK [6]	NO
	FlowTags [7]	NO
	SIMPLE [8]	NO
	OpenSAFE [9]	NO
	CloudWatcher [10]	NO
SDN for security as a service		
Spoofing detection	Du et al. [13]	NO
	BASE [14]	NO
	SEFA [17]	NO
Scanning and anomaly detection	MTD [18]	NO
	AnonyFlow [21]	NO
	L-IDS [22]	YES
	Mehdi et al. [23]	NO
DoS/DDoS mitigation	Braga et al. [29]	YES
	Dillon [33]	NO
Forensic techniques	Adam et al. [34]	NO
Security framework	FRESCO [36]	NO

6 Conclusion

SDNs are known for their resilience and antifragile nature. This paper has outlined a model for securing information from cyber threats in capacity of SDNs in addition to introducing the concept of antifragile cyber security. Our proposed work is a prelude to this concept and serves as the basis for thinking differently in terms of cyber security in this fast pacing era of technology. SDNs are currently trending as the enabler for security; however, requirement for SDN's own security cannot be denied. This paper has highlighted the contributions made for SDN-enabled security and their potential for making systems more than resilient. Securing SDN itself is the basic necessity and this paper has argued on bridging the gap between both "Security with SDN" and "Security for SDN" by presenting a unified model of Antifragile Cyber Security within SDN. The model has also highlighted the major cyber threats prevailing in the SDN-based systems and has indicated the focal points of concern whether information is being processed by the application, transiting the network, or resident in storage.

7 Future work

The proposed model can be implemented using SDN framework, focusing on exploiting the benefits of SDN in terms of its antifragile nature. The future plan is to contribute to the same model by practically implementing it as suite of tools to judge the efficiency of our proposed idea of antifragile cyber security.

References

1. Antifragile-Wikipedia. https://en.wikipedia.org/wiki/Antifragile. Accessed 02 Oct 2015
2. SDN Architecture, Issue 1, June 2014. https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf. Accessed 16 June 2015
3. Scott-Hayward S, O'Callaghan G, Sezer S (2013) Sdn security: a survey. Future Networks and Services (SDN4FNS), 2013 IEEE SDN for, vol. no, pp 1, 7, 11–13 Nov 2013 doi:10.1109/SDN4FNS.2013.6702553
4. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Turner J (2008) OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput Commu Rev 38(2):69–74
5. Schehlmann L, Abt S, Baier H (2014) Blessing or curse? Revisiting security aspects of Software-Defined Networking. 2014 10th international conference on network and service management (CNSM)
6. Anwer B, Benson T, Feamster N, Levin D, Rexford J (2013) A slick control plane for network middleboxes. In: Proceedings of the second ACM SIGCOMM workshop on hot topics in software defined networking, pp 147–148. ACM
7. Fayazbakhsh SK, Sekar V, Yu M, Mogul JC (2013) Flowtags: enforcing network-wide policies in the presence of dynamic middlebox actions. In: Proceedings of the second ACM SIGCOMM workshop on hot topics in software defined networking, pp 19–24. ACM
8. Qazi ZA, Tu CC, Chiang L, Miao R, Sekar V, Yu M (2013) SIMPLE-fying middlebox policy enforcement using SDN. In: ACM SIGCOMM computer communication review, vol 43, no 4, pp. 27–38. ACM
9. Ballard JR, Rae I, Akella A (2010) Extensible and scalable network monitoring using opensafe. Proc, INM/WREN
10. Shin S, Gu G (2012) CloudWatcher: network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In: 2012 20th IEEE international conference on network protocols (ICNP), pp 1–6. IEEE
11. Gude N, Koponen T, Pettit J, Pfaff B, Casado M, McKeown N, Shenker S (2008) NOX: towards an operating system for networks. ACM SIGCOMM Comput Commun Rev 38(3):105–110
12. Sniedovich M (2006) Dijkstra's algorithm revisited: the dynamic programming connexion. Control Cybern 35(3):599
13. Du X, Wang MZ, Zhang X, Zhu L (2014) Traffic-based Malicious Switch Detection in SDN. Int J Secur Its Appl 8(5):119–130
14. Kwon J, Seo D, Kwon M, Lee H, Perrig A, Kim H (2015) An incrementally deployable anti-spoofing mechanism for software-defined networks. Comput Commun
15. Yaar A, Perrig A, Song D (2003) Pi: a path identification mechanism to defend against DDoS attacks. In: Proceedings. 2003 Symposium on security and privacy, 2003, pp 93–107. IEEE
16. Park K, Lee H (2001) On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. In: ACM SIGCOMM computer communication review, vol 31, no 4, pp 15–26. ACM
17. Yao G, Bi J, Feng T, Xiao P, Zhou D (2014) Performing software defined route-based IP spoofing filtering with SEFA. In: 2014 23rd international conference on computer communication and networks (ICCCN), pp 1–8. IEEE
18. Jafarian JH, Al-Shaer E, Duan Q (2012) Openflow random host mutation: transparent moving target defense using software defined networking. In: Proceedings of the first workshop on hot topics in software defined networks, pp 127–132. ACM
19. NP-Hardness-Wikipedia. https://en.wikipedia.org/wiki/NP-hardness. Accessed 07 June 2015
20. Satisfiability Modulo Theories-Wikipedia. https://en.wikipedia.org/wiki/Satisfiability_modulo_theories. Accessed 07 June 2015
21. Mendonca M, Seetharaman S, Obraczka K (2012) A flexible in-network IP anonymization service. In: 2012 IEEE international conference on communications (ICC), pp 6651–6656. IEEE
22. Skowyra R, Bahargam S, Bestavros A (2013) Software-defined IDS for securing embedded mobile devices. High performance extreme computing conference (HPEC), 2013 IEEE, vol. no, pp 1, 7, 10–12 Sept. 2013
23. Mehdi SA, Khalid J, Khayam SA (2011) Revisiting traffic anomaly detection using software defined networking. Recent advances in intrusion detection. Springer, Berlin Heidelberg, pp 161–180
24. Schechter SE, Jung J, Berger AW (2004) Fast detection of scanning worm infections. In: RAID. pp 59–81
25. Twycross J, Williamson MM (2003) Implementing and testing a virus throttle. Proceedings of the 12th conference on USENIX security symposium, vol 12. USENIX Association, Berkeley, CA, USA, pp 20–20
26. Williamson MM (2002) Throttling viruses: restricting propagation to defeat malicious mobile code. In: Proceedings of 18th annual computer security applications conference, 2002, pp 61–68. IEEE
27. Gu Y, McCallum A, Towsley D (2005) Detecting anomalies in network traffic using maximum entropy estimation. In: Proceedings of the 5th ACM SIGCOMM conference on internet measurement, pp 32–32. IMC '05, USENIX Association, Berkeley, CA, USA
28. Mahoney MV (2003) Network traffic anomaly detection based on packet bytes. In: Proceedings of the 2003 ACM symposium on applied computing, pp 346–350. SAC'03, ACM, New York, NY, USA
29. Kullback Leibler Divergence-Wikipedia. https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence
30. Braga R, Mota E, Passito A (2010) Lightweight DDoS flooding attack detection using NOX/OpenFlow. 2010 IEEE 35th conference on local computer networks (LCN), vol. no, pp 408, 415, 10–14 Oct 2010. doi:10.1109/LCN.2010.5735752
31. Kohonen T (1990) The self-organizing map. Proc IEEE 78(9):1464–1480
32. KDD Cup 1999: Computer Network intrusion Detection. [Online] http://www.sigkdd.org/kdd-cup-1999-computer-network-intrusion-detection
33. Dillon C, Berkelaar M OpenFlow (D)DoS nitigation. [Online] http://www.delaat.net/rp/2013-2014/p42/report.pdf
34. Bates A, Butler K, Haeberlen A, Sherr M, Zhou W (2014) Let SDN be your eyes: secure forensics in data center networks. In: Proceedings of the NDSS workshop on security of emerging network technologies (SENT'14)
35. Haeberlen A, Kouznetsov P, Druschel P (2007) PeerReview: practical accountability for distributed systems. In: ACM SIGOPS operating systems review, vol 41, no. 6, pp 175–188. ACM
36. Shin S, Porras PA, Yegneswaran V, Fong MW, Gu G, Tyson M (2013) Modular composable security services for software-defined networks, FRESCO. In: NDSS
37. SRI International. BotHunter: a network-based Botnet diagnosis system. http://www.bothunter.net/
38. Kreutz D, Ramos F, Verissimo P (2013) Towards secure and dependable software-defined networks. In: Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, pp 55–60. ACM

39. The STRIDE Threat Model https://msdn.microsoft.com/en-us/library/ee823878
40. Schneier B Attack trees: modeling security threats. Dr. Dobb's journal (1999)
41. Saini V, Duan Q, Paruchuri V (2008) Threat modeling using attack trees. J Comput Sci Coll 23(4):124–131
42. Kloti R, Kotronis V, Smith P (2013) OpenFlow: a security analysis. 2013 21st IEEE international conference on network protocols (ICNP), vol. no, pp 1, 6, 7–10 Oct 2013. doi:10.1109/ICNP.2013.6733671
43. Shin S, Gu G (2013) Attacking software-defined networks: a first feasibility study. In: Proceedings of the second ACM SIGCOMM workshop on hot topics in software defined networking, pp 165–166. ACM
44. Hong S, Xu L, Wang H, Gu G (2015) Poisoning network visibility in software-defined networks: new attacks and countermeasures. NDSS
45. Al-Shaer E, Al-Haj S (2010) FlowChecker: configuration analysis and verification of federated OpenFlow infrastructures. In: Proceedings of the 3rd ACM workshop on assurable and usable security configuration, pp 37–44. ACM
46. Porras P, Shin S, Yegneswaran V, Fong M, Tyson M, Gu G (2012) A security enforcement kernel for OpenFlow networks. In: Proceedings of the first workshop on hot topics in software defined networks, pp 121–126. ACM
47. Canini M, Venzano D, Peresini P, Kostic D, Rexford J (2012) A NICE way to test OpenFlow applications. NSDI 12:127–140
48. Khurshid A, Zhou W, Caesar M, Godfrey P (2012) Veriflow: verifying network-wide invariants in real time. ACM SIGCOMM Comput Commun Rev 42(4):467–472
49. Mininet. http://mininet.org/
50. Yan Z, Prehofer C (2011) Autonomic trust management for a component-based software system. Dependable Secure Comput IEEE Trans 8(6):810–823
51. Wen X, Chen Y, Hu C, Shi C, Wang Y (2013) Towards a secure controller platform for openflow applications. In: Proceedings of the second ACM SIGCOMM workshop on hot topics in software defined networking, pp 171–172. ACM
52. http://www.bizforum.org/whitepapers/candle-4.htm
53. Threat Model-Wikipedia. http://en.wikipedia.org/wiki/Threat_model. Accessed 09 Feb 2015
54. Octotrike Threat Model. http://octotrike.org/. Accessed 10 Feb 2015
55. http://shop.standards.co.nz/default.htm?url=webshop/&action=viewSearchProduct&mod=catalog&pid=4360:2004%28AS|NZS%29
56. CVSS. https://nvd.nist.gov/cvss.cfm
57. Octave. http://www.cert.org/resilience/products-services/octave/index.cfm
58. http://is355.wikidot.com/information-security-general
59. Fiorini RA, Santacroce GF (2015) Application resilience and antifragility from the internet of medical devices to Healthcare Governance Systems. EJBI 11(3):
60. Abid A, Khemakhem MT, Marzouk S, Jemaa MB, Monteil T, Drira K (2014) Toward antifragile cloud computing infrastructures. Procedia Comput Sci 32:850–855
61. De Florio V (2014) Antifragility = elasticity + resilience + machine learning models and algorithms for open system fidelity. Procedia Comput Sci 32:834–841
62. Jones KH (2014) Engineering antifragile systems: a change in design philosophy. Procedia Comput Sci 32:870–875
63. De Florio V (2015) On resilient behaviors in computational systems and environments. J Reliab Intell Env 1–14
64. Nayak AK, Reimers A, Feamster N, Clark R (2009) Resonance: dynamic access control for enterprise networks. In: Proceedings of the 1st ACM workshop on research on enterprise networking, pp 11–18. ACM
65. Naous J, Stutsman R, Mazieres D, McKeown N, Zeldovich N (2009) Delegating network security with more information. In: Proceedings of the 1st ACM workshop on research on enterprise networking, pp 19–26. ACM
66. Rogers DM (2005) Anti-forensic presentation given to Lockheed Martin. San Diego