2014

OpenSec: A Framework for Implementing Security Policies using OpenFlow

Adrian Lara
University of Nebraska-Lincoln, alara@cse.unl.edu

Byrav Ramamurthy
University of Nebraska-Lincoln, bramamurthy2@unl.edu

Globecom 2014 - Communication and Information System Security Symposium
Pages: 781 - 786, DOI: 10.1109/GLOCOM.2014.7036903

OpenSec: A Framework for Implementing Security Policies using OpenFlow

Adrian Lara Byrav Ramamurthy
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588-0115, USA
Email: {alara,byrav}@cse.unl.edu

***Abstract*—As the popularity of software defined networks (SDN) and OpenFlow increases, policy-driven network management has received more attention. Manual configuration of multiple devices is being replaced by an automated approach where a software-based, network-aware controller handles the configuration of all network devices. Software applications running on top of the network controller provide an abstraction of the topology and facilitate the task of operating the network.**

We propose OpenSec, an OpenFlow-based security framework that allows a network security operator to create and implement security policies written in human-readable language. Using OpenSec, the user can describe a flow in terms of OpenFlow matching fields, define which security services must be applied to that flow (deep packet inspection, intrusion detection, spam detection, etc) and specify security levels that define how OpenSec reacts if malicious traffic is detected. We implement OpenSec in the GENI testbed to evaluate the flexibility, accuracy and scalability of the framework. The experimental setup includes deep packet inspection, intrusion detection and network quarantining to secure a web server from network scanners. We achieve a constant delay when reacting to security alerts and a detection rate of 98%.

***Index Terms*—Software Defined Networking, OpenFlow, Network Security**

I. Introduction

With the advent of software-defined networks, efforts to automate and simplify network operation have become popular [1], [2], [3]. In SDN, the complexity of the network shifts towards the controller and brings simplicity and abstraction to the network operator. As we move away from manual configuration at each device, we get closer to automated implementation of network policies and rules. SDN decouples the control plane from the data plane and migrates the former to a logically centralized software-based network controller. More complex network-control applications can thus be implemented at the controller and exploit the fact that they are network-wide aware due to the centralized nature of the control plane.

OpenFlow is a protocol that standardizes how an SDN controller communicates with the network devices. An OpenFlow-compliant switch exposes to the controller an abstraction of its flow table and allows the controller to manipulate it by inserting, modifying or deleting rules in the table. Using OpenFlow, an application running on the network controller can thus control how one or more layer 2 switches forward incoming packets.

In this paper, we propose OpenSec, an OpenFlow-based network security framework that allows operators to implement security policies across the network. Because OpenSec provides an abstraction of the network, the operators can focus on designing simple and human-readable security policies, instead of on configuring all the devices to achieve the desired security. OpenSec consists of a software layer running on top of the network controller and multiple external devices that perform security services (such as firewall, encryption, spam detection, deep packet inspection (DPI) and others) and report the results to the controller. The main goal of OpenSec is to allow network operators to describe security policies for specific flows. The policies include a description of the flow, a list of security services that apply to the flow and how to react in case malicious content is found. The reaction can be to alert only, or to quarantine traffic and even to fully block all packets from a specific source.

We have built OpenSec taking two design requirements into consideration. First, policies should be human-readable. Simplicity is one of the main goals of our framework and although current work has focused on creating human-readable policies [4], [2], [5], we argue that there is still room for improvement. Second, the majority of the workload should be done by the processing units. When the controller is responsible for all tasks it becomes a bottleneck and the solution does not scale well. In OpenSec, the controller is subject to a low workload and is responsible of implementing policies and modifying forwarding rules based on the security alerts received from the processing units.

To evaluate the framework, we implement it on the GENI testbed [6] using virtual nodes and Open vSwitch instances. We implement two processing units that perform DPI and intrusion detection and we evaluate the performance of using different policies to secure a web server from a port scanner. By combining both units, we achieve a detection rate of 98%

This work was supported in part by an NSF FIA-NP grant (CNS-1345277).

978-1-4799-3512-3/14/$31.00 ©2014 IEEE

and a constant-time delay to react to security alerts issued by the processing units.

Our contributions in this paper are as follows:

1) We create a simple, human-readable language to automatically implement network security policies.
2) We give a first step towards automated, policy-based reaction to security alerts using OpenFlow.

The rest of this paper is organized as follows. In section II we discuss required background on OpenFlow and related work. In section III we describe OpenSec and in section IV we evaluate the framework using a working prototype. Finally we conclude in Section V.

II. Background: SDN and OpenFlow

Software Defined Networking (SDN) consists of decoupling the control from the data plane. Forwarding devices become simpler and transmit packets based on the forwarding table, which is manipulated by a software-based, logically-centralized network controller. SDN-based networks have several capabilities that can be exploited in the context of network security [7]. First, the controller is network-aware. This allows it to gather information from multiple locations of the network and to react accordingly. Second, SDN greatly simplifies dynamic updating of traffic rules. Software running on the controller can automatically modify the forwarding table of any network device, based on the observation of current traffic.

OpenFlow [8] is the most commonly deployed SDN protocol. It standardizes the communication between a software-based controller and layer 2 switches through the OpenFlow channel. An OpenFlow-compliant switch exposes an abstraction of its forwarding table to the OpenFlow controller. The controller can insert, delete or modify flows to the forwarding table of any device. Each flow in the table consists of match fields, an action (forward to a port, drop, translate VLAN tag, etc.) and statistic data about the flow. When a packet received by a switch does not match any rule, it is forwarded to the controller. The controller can listen to incoming packets, push outgoing packets and push new rules to the flow table of a switch. OpenFlow 1.0 provides the following match fields: ingress port, Ethernet source, destination and type, VLAN id, VLAN priority, IP source, destination, protocol and ToS bits and TCP/UDC source and destination.

Next we describe how OpenFlow has been used to enable policy-based network administration and also how OpenFlow has been use to provide network security.

A. Network policies using OpenFlow

Foster et al. [9] propose Frenetic, a programming language to program OpenFlow-based networks. Frenetic provides an interface to query traffic information and to react to network events. Frenetic focuses on simplifying how to program network events and how to retrieve traffic information. OpenSec focuses on hiding such complexity and allowing a security operator to work at a higher level. The goal of OpenSec is to automate the implementation of security policies, not to simplify the programming of the network.

More recently, Voelli et al. [4] proposed Procera, a "functional reactive programming" framework where a user can write a high-level policy to define how to handle network events. Just like Frenetic, Procera also aims to simplifying how to deal with network events. OpenSec also relies on defining network policies, but we focus more on implementing the policy and reacting automatically, instead of providing a network-programming interface to the user.

B. Network security using OpenFlow

Shin et al. [5] propose CloudWatcher, a security monitoring framework for the cloud that has several similarities with our work. Using CloudWatcher, a network operator can use a policy to describe a flow and describe which security services must be applied it. The authors focus on routing algorithms to locate processing units. Cloudwatcher focuses more on routing algorithms to locate the processing units, while we focus more on the high-level policies and automated reaction.

FRESCO [10] is another OpenFlow-based security framework that exposes security modules to external users, who can in turn define security polices using such modules. The main difference between FRESCO and OpenSec is that FRESCO performs all the security processing in the controller, whereas OpenSec uses external processing units that send information to the controller.

Although other studies have addressed policy-based network administration using OpenFlow, as well as providing security through SDN, OpenSec's innovative approach allows operators to customize the security of the network using human-readable policies and to customize how the controller reacts automatically when malicious traffic is detected. We describe our proposed framework in the following section.

III. The OpenSec framework

OpenSec aims at allowing network operators to create and implement network security policies. The policies include a description of the flow, a set of security services that must be applied to such traffic and a security level for automatic reaction in case of detecting malicious traffic. The processing units provide specific security services such as encryption, denial of service attack detection, deep packet inspection or any other.

A. Policy parser

The policy parser converts the policy definition file into data and structures that can be processed by OpenSec. Policies can be defined using the keywords shown in Table I.

One of the main goals of OpenSec is to allow operators to create very simple policies to control the network. Among all related work, Procera is probably the one that has focused

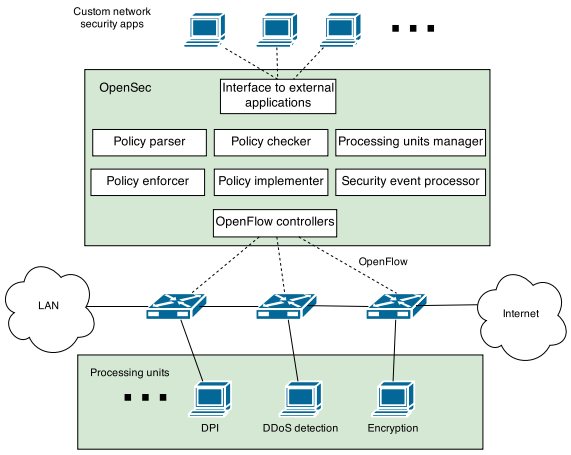

Fig. 1. The OpenSec framework.

TABLE I
SYNTAX TO CREATE POLICIES USING OPENSEC

	Value	Description
Flow	VLAN, macSrc, macDst, macInPort, macOutPort, ipSrc, ipDst, l4Port	Uses OpenFlow match fields to describe a flow
Service	Encrypt, IDS, DPI, spam, DDoS or any other service registered	Identifies a security service that should be applied to the flow
React	notify-only, quarantine, block	Determines how to react if the service reports malicious content

most on designing human-readable policy specification. However, we argue that understanding a definition written in Procera is not straightforward. The following instructions define a rule that allows all traffic:
proc *world* $\rightarrow$ **do**; **returnA:** λ *req* $\rightarrow$ *allow*

This statement still contains symbols that make it complicated to read. Instead, we aim at statements such as:
Flow: VLAN=192; **Service:** DPI; **React:** alert-only

Procera relies on reactive programming and its goal is different from OpenSec. In Procera, the goal is to program the network using policies and this includes handling events generated by the switches. OpenSec does not communicate the network events to the end-user. Instead, they are automatically processed. This allows us to use a much simpler syntax to describe the flow, identify one or more services and specify how to react when malicious traffic is detected.

B. Policy checker

The policy checker maintains information about all policies currently enforced and checks that new policies do not conflict with existing ones. For example, if two policies use exactly the same matching fields and the same values, implementing such policy will not work, since one of the two matches will occur first. By keeping a reference to all policy objects in the system, the checker can easily compare incoming policies with existing ones to verify this.

C. Processing units manager

In earlier work [3] we described how specialized network hardware (commonly known as middleboxes) should not be located in choke points of the topology, traversed by all traffic. Instead, these devices should be located outside of the main path and should act as security processing units that are visited only by the traffic that needs to be processed. Using a smarter OpenFlow-based control plane, the network can dynamically create rules to re-route traffic only when necessary. In Fig. 1, note that the main path is between the LAN and the WAN and there are only simple, OpenFlow-compliant switches on that path. This is important in terms of performance and reliability, since a L2 switch is easier to maintain, upgrade or replace in comparison to specialized hardware. When a specific flow is subject to deep packet inspection, for example, then the controller adds a rule that forces such traffic to visit the DPI processing unit.

TABLE II
REGISTERED SECURITY PROCESSING UNITS FOR FIG. 1.

Service type	Switch ID	In-interface	Out-interface
DPI	1	25	26
DDoS	2	48	49
Encrypt	3	25	26

In order for this architecture to function, all processing units must register with OpenSec. The processing units manager collects all the registrations and ends up with a list of services and the location in the network where they can be found. In our current implementation, the controller maps a service id (DPI, IPS) to a switchID, an input port and an output port. This is all the data needed by OpenSec to manipulate the flow table of the devices in order to re-route traffic to the processing units.

D. Policy implementer

Implementing a policy implies re-routing matching traffic to one or more security processing units. In our current implementation, OpenSec first queries the processing units manager for the switch id where the processing unit is connected. In that switch, a rule is pushed to duplicate traffic so that a copy is sent to the input interface of the processing unit for further processing.

Our current implementation assumes that rules to forward traffic from the source to the destination already exist. Therefore, OpenSec queries the flow table of the device attached to a security service unit and finds the rule that matches the description of the flow. Finally, that flow is updated as needed.

E. Security event processor

One of the most important features of OpenSec is the automatic reaction to security alerts. Usually, a network operator will react to an alert by either ignoring it or blocking the source of the suspicious traffic. In OpenSec, the network operator can define such a reaction in advance using three possible solutions: *alert, quarantine* or *block*.

To allow OpenSec to scale better, the processing units are responsible for analyzing the traffic and detecting malicious flows. Sampling traffic at the controller increases the chances of having a bottleneck and increases the complexity. Also, the connectivity between switches and controller is usually of low bandwidth, since it is the control plane. In contrast, the data plane allows a faster bit rate and the processing units are optimized to handle big flows. In OpenSec, the controller remains listening to alerts and reacts to those alerts by deciding how to modify the traffic rules.

If the specified reaction is **alert**, forwarding rules are not modified and the network administrator is notified by e-mail. To quarantine traffic, a processing unit logging all traffic is attached to one of the switches. OpenSec updates the forwarding table of the switch so that matching traffic is forwarded to the quarantine unit instead of to the host. The quarantine unit logs all incoming traffic so that a network operator can then analyze the data. Finally, if the policy requires blocking all traffic, then the forwarding rules of involved switches are modified so that matching traffic is dropped.

To summarize this section, we provide a step-by-step example of how an operator can implement a policy using OpenSec.

F. Step-by-step example

Suppose that we implement the following policy using the sample network in Fig. 1:
Flow: VLAN=192
Service: DPI
React: block.

To implement the policy:

1) A network admin should write the policy in a file (in our current implementation, OpenSec will read policies from a given folder).
2) OpenSec parses the policy and locates the switch where the DPI unit is connected, as well as the interface (switch 1, port 25 as shown in Table II).
3) OpenSec assumes that there one or more rules already exist so that traffic from VLAN 192 can go through the network.
4) OpenSec finds the rule in switch 1 that matches packets tagged with VLAN 192.
5) OpenSec modifies the rule so that traffic is forwarded as specified by the original rule, but also forwarded to port 25.

The processing unit can be implemented in any way, as long as it is capable of registering with the controller and issuing alerts. The reaction to an alert issued by the processing unit is completely automatic and needs no intervention from the network operator.

1) The processing unit detects malicious traffic and sends a notification to the controller, with a description of the flow.
2) OpenSec finds the match that corresponds to the description provided by unit and locates the corresponding policy.
3) Because the policy specifies that traffic should be blocked, the flow rule that was originally modified is updated to drop all packets.

Next we describe evaluation results. We implemented OpenSec in the GENI testbed to evaluate the scalability, flexibility and accuracy of the framework.

IV. Evaluation

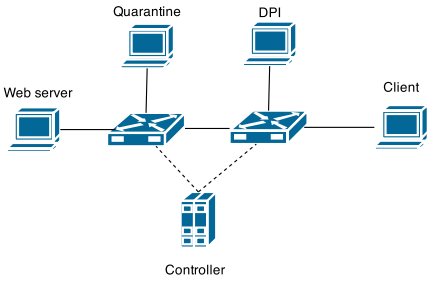

Fig. 2. Sample topology with DPI and quarantining.

In Fig. 2, a testing client is connected to a web server through two Open vSwitch nodes. The goal of the experiment is protect the web server from port scanners. The experimental setup consists of sending malicious and normal traffic to the web server. Normal traffic is created by replaying commonly browsing requests such as video streaming, web requests and file downloads. A total of 20,000 packets are used. Malicious traffic is generated using nmap [11], a port scanning tool. Specifically, we run the following attacks: SYN stealth scan, Xmas tree scan, ping scan, UDP scan and TCP scan [12]. These scans send 2,000 packets each and scan multiple ports to explore which protocols are exposed to the network. In this secion we discuss how OpenFlow can be used to protect the web server from these scans.

A. Deep packet inspection on all incoming packets

First, we experiment using a broad policy that sends all incoming traffic to a deep packet inspection unit. The policy is as follows:
Policy 1: **Flow:** inPort=1; **Service:** DPI; **React:** quarantine.

This policy ensures that all traffic coming through port 1 is duplicated and sent to the DPI unit. It also specifies that, in case of detecting malicious traffic, all source coming from the suspected host should be sent to quarantine. The quarantine unit listens to the network interface and records all contents, so that traffic samples can be analyzed afterwards.

The DPI unit is connected to the right-most switch and the quarantine unit is located to the left-most switch. Our DPI unit is built on top of nDPI [13], an open source DPI tool. nDPI supports all major networking protocols at any layer, such as IPv4, IPv6, UDP, TCP, HTTP, DNS, SSH, SMTP, Flash and many others. A complete list is availble at the website [13]. Using the nDPI library, we analyze traffic samples captures using pcap every second. The implemented tool detects the number of flows, the number of packets within a flow and the application-layer protocol used by each flow.

The tool classifies traffic as malicious if: 1) the application-layer protocol is unknown to nDPI, 2) there are more than five flows with only one or two packets per flow and 3) if the same source uses more than five different application-layer protocols within a five seconds sample interval. If malicious traffic is detected, the unit sends a simple message to the OpenFlow controller containing the description of the flow. The controller can then take action based on the reaction specified by the policy for such a flow.

B. Scalability

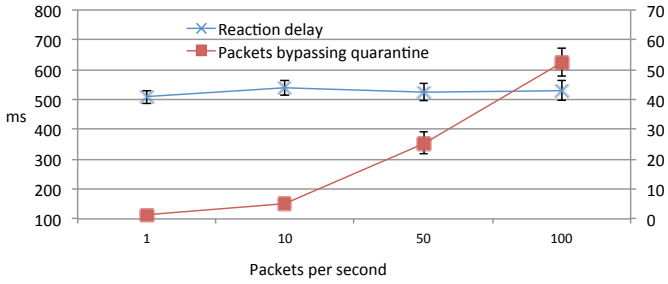

Fig. 3. Delay needed to quarantine traffic and number of packets that bypass the quarantine.

OpenSec scales well because only the processing units (instead of the controller) deal with the increasing amount of traffic. In the DPI example, the processing unit is capable of inspecting traffic at high bit rates, but the load at the controller remains minimum. As shown in Fig. 3, the delay needed to quarantine malicious traffic remains constant independently of the number of packets per second. This delay remains constant because it does not depend on the number of attacks detected or the packet arrival rate. For every alert, the controller simply finds the matching policy and modifies the traffic rules as required. Also, if the capabilities of the processing units must be improved, this task is independent of OpenSec and can be performed without modifying the controller. As we discuss in earlier work [3], this framework is easier to deploy in comparison to when a middlebox is located in the main data path.

Due to the fact that the delay remains constant, the number of packets that bypass the quarantine unit grows linearly. Since some time elapses between the detection of the scan and the time when the flow rules are modified, a certain number of packets are still forwarded to the web server instead of the quarantine unit. This result is also shown in Fig. 3. The policy deployed for this prototype works well to detect attacks that are carried over multiple packets, such as a denial of service attack. In such scenarios, reacting to the attack after a small number of packets have reached the server is acceptable. If the requirement is to detect smaller attacks that are carried over a small number of packets (SQL injection, for example), then a different policy can send all traffic to the unit and allow the unit to drop packets in real time.

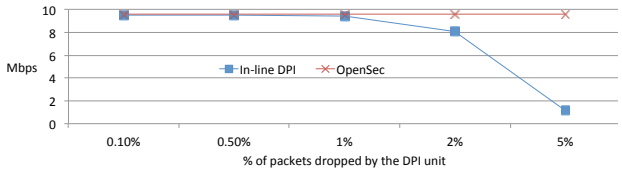

Fig. 4. TCP throughput achieved using OpenSec and in-line DPI using a 10Mbps link.

C. Duplicating traffic or in-line DPI

One could argue that 50 packets bypassing the quarantine is not fast enough. One way to solve this could be to use a traditional approach where the DPI unit is traversed by traffic, allowing the tool to drop packets in real time. However, security middleboxes such as firewalls and DPI units are not yet capable of dealing with big data flows without dropping packets. Dart et al. [14] show the TCP throughput can be reduced by a factor of 9 if a router is losing a ver small percentage of traffic. To verify this, we simulated a DPI unit that is traversed by all traffic and then sends all data back to the main data path. We experimented with different percentages of packet drops and Fig. 4 shows how this solution heavily impacts the TCP throughput between the server and the client. By duplicating the traffic, OpenSec increases the amount of traffic, but also allows the data to traverse the network at a faster rate.

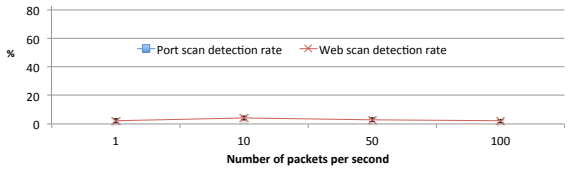

Fig. 5. Accuracy achieved by DPI only.

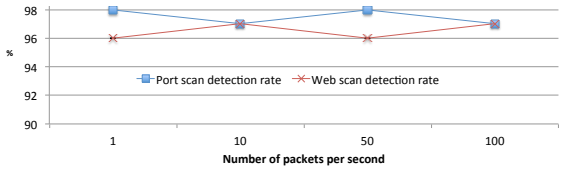

Fig. 6. Accuracy achieved by DPI and IDS together.

D. IDS processing for HTTP specific traffic

Figure 5 shows a high detection rate for nmap port scans when policy 1 is used. Although our nDPI script is fairly simple, it is highly accurate to detect port scans, since the number of scanned ports increases almost immediately. However, when we add web scans to our experiment, we notice that the detection rate is very low.

We use Arachni [15] to run the scans. The tool sends approximately 300 requests to commonly used resources within the webserver searching for vulnerabilities and backdoors. As shown in Fig. 5, the nDPI script has a very low detection rate for this type of scan. This is expected because Arachni sends all traffic through TCP/80 and nDPI does not have any HTTP-specific capability to know that this is a scan.

Instead of increasing the complexity of the DPI unit (which is not the goal of this paper), next we focus on showing how OpenSec allows for a more flexible security management. We add a second processing unit that uses Bro [16], an open source intrusion detection system (IDS) tool. We create a customized script in Bro to extract the URL from each request and issue an alert if the same host queries more than 20 different URLs in less than a second. Using OpenSec, we add the following policy to the previous one:
Policy 2: **Flow:** l4Port=80; **Service:** IDS; **React:** quarantine.

Figure 6 shows how the detection rate of web scans is strongly increased. By simply adding a new policy that only affects TCP/80 traffic, a detection rate of 98% is achieved.. The IDS script ignores all packets that are not HTTP content. Therefore, in a larger network where web traffic is only a portion of the load, policy 2 is useful to send only HTTP traffic to the IDS unit instead of sending all the remaining unnecessary data.

V. Conclusion

In this paper we present OpenSec, an OpenFlow-based framework that allows network operators to describe security policies using human-readable language and to implement them across the network. OpenSec acts as a virtual layer between the user and the complexity of the OpenFlow controller and automatically converts security policies into a set of rules that are pushed into network devices. OpenSec also allows network operators to specify how to automatically react in case of detecting malicious traffic. OpenSec is the allows for automated reaction to security alerts based on pre-defined network policies. By doing so, it contributes to hiding the complexity of the network to security operators, who only need to focus on defining the policies.

References

[1] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, October 2007.
[2] H. Kim and N. Feamster, "Improving network management with software defined networking," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 114–119, February 2013.
[3] A. Lara, A. Kolasani, and B. Ramamurthy, "Simplifying network management using Software Defined Networking and OpenFlow," in *2012 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, Bangalore, India, December 2012, pp. 24–29.
[4] A. Voellmy, H. Kim, and N. Feamster, "Procera: a language for high-level reactive network control," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, Helsinki, Finland, August 2012.
[5] S. Shin and G. Gu, "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)*, Austin, Texas, October 2012, pp. 1–6.
[6] GENI Portal. [Online]. Available: http://portal.geni.net
[7] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 493–512, First Quarter 2014.
[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.
[9] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: a network programming language," in *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, Tokyo, Japan, September 2011.
[10] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "FRESCO: Modular Composable Security Services for Software-De ned Networks," in *Internet Society Network and Distributed System Security Symposium (NDSS)*, San Diego, California, U.S.A., February 2013.
[11] nmap security scanner. [Online]. Available: www.nmap.org
[12] A. J. Bennieston. NMAP - A Stealth Port Scanner. [Online]. Available: http://nmap.org/bennieston-tutorial/
[13] nDPI: Open and Extensible LGPLv3 Deep Packet Inspection Library. [Online]. Available: http://www.ntop.org/products/ndpi/
[14] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The Science DMZ: A Network Design Pattern for Data-intensive Science," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 85:1–85:10.
[15] Arachni: A web application security scanner framework. [Online]. Available: http://www.arachni-scanner.com/
[16] The Bro Network Security Monitor. [Online]. Available: www.bro.org