The vAMP Attack:

Taking Control of Cloud Systems via the Unified Packet Parser

Kashyap Thimmaraju
TU Berlin
kash@sec.t-labs.tu-berlin.de

Bhargava Shastry
TU Berlin
bshastry@sec.t-labs.tu-berlin.de

Tobias Fiebig
TU Berlin
tobias@inet.tu-berlin.de

Felicitas Hetzelt
TU Berlin
file@sec.t-labs.tu-berlin.de

Jean-Pierre Seifert
TU Berlin
jpseifert@sec.t-labs.tu-berlin.de

Anja Feldmann
TU Berlin
anja@inet.tu-berlin.de

Stefan Schmid
TU Berlin/Aalborg University
schmiste@cs.aau.dk

ABSTRACT

Virtual switches are a crucial component of cloud operating systems that interconnect virtual machines in a flexible manner. They implement complex network protocol parsing in the *unified packet parser*—parsing all supported packet header fields in a single pass—and are commonly co-located with the virtualization layer. We find that this significantly reduces the barrier for low-budget attackers to launch high impact attacks in the cloud. This leads us to introduce the *virtual switch attacker model for packet-parsing*, in short the *vAMP attack*. Using OpenStack, a cloud operating system, and Open vSwitch, a virtual switch, we demonstrate how current virtual switch designs cannot withstand vAMP. Thereby giving a weak attacker full control of the cloud in a matter of minutes.

KEYWORDS

Network Isolation; Network Virtualization; Data Plane Security; Packet Parsing; MPLS; Virtual Switches; Open vSwitch; Cloud Security; OpenStack; Attacker Models; ROP; SDN; NFV

CCSW'17, November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-5204-8/17/11…$15.00
https://doi.org/10.1145/3140649.3140651

1 INTRODUCTION

Computer networks are becoming increasingly programmable and virtualized. A key enabler for such a paradigm is the virtual switch. It is a piece of software that resides in the server's virtualization layer (e.g., VMware's ESXi hypervisor or Xen's Dom0), tasked with virtualizing the data plane of virtual machines. Hence, virtual switches are meant to provide *network isolation* among tenant's virtual machines [1].

Virtual switches such as Open vSwitch (OvS), Cisco Nexus 1000V, VMware vSwitch, and Microsoft VFP, bring new flexibility to data centers in terms of network virtualization [1], e.g., centralized control of all virtual switches, and layer 2 virtual private networks.

Despite their popularity, the security implications of virtual switches have received little attention. In general, while much research has focussed on control plane security, the security of the physical and virtual data plane is often overlooked. This is worrisome as the data plane in general, and packet parsers in particular process attacker controlled input.

This paper is motivated by two key observations. First, by placing virtual switches in the (edge) servers, the attack surface has advanced one step closer to the attacker. Therefore, associated attacker models may significantly change compared to non-virtualized data planes. Second, unified packet parsing—parsing *all* the supported protocol fields of a packet, e.g., Ethernet, IP and TCP, in a single pass instead of several independent passes—exacerbates the existing attack surface of integrating, and juxtaposing complex network functionality with the virtual switch. In fact, the number of parsed protocols for virtual switches appears to be growing constantly as depicted in Fig. 1. The increasing complexity

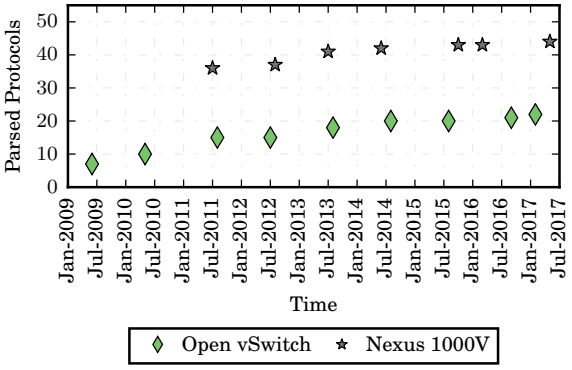

Figure 1: The total number of parsed high-level protocols in two popular virtual switches from 2009-2017.

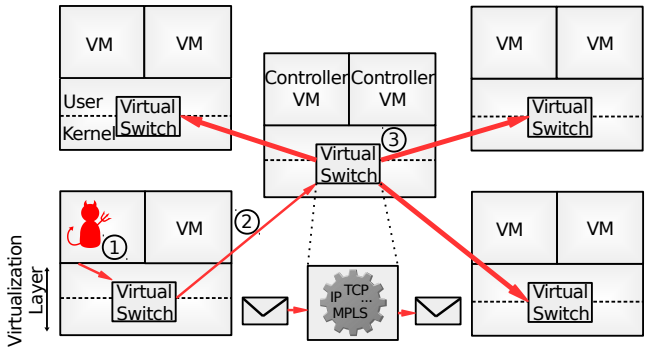

Figure 2: As a consequences of the vAMP attack, a worm can propagate to all the systems by exploiting the virtual switch's unified packet parser.

of packet parsers lowers the bar for adversaries to attack [2] cloud systems where virtual switches are commonly used.

These observations lead us, in this paper, to introduce a new attacker model for virtual switches. The virtual switch attacker model for packet-parsing, in short: the *vAMP attack*, describes a low-resource attacker with a huge impact.

We demonstrate this attacker model, in a case study using OpenStack, a popular cloud operating system, and OvS as the virtual switch. Using an off-the-shelf fuzzer to fuzz the unified packet parser of OvS, which is merely 2-3% of the OvS code base, we uncovered multiple security vulnerabilities.

Exploiting just one of those vulnerabilities enabled us to subsequently compromise the entire OpenStack setup, as illustrated in Fig. 2. An attacker sends a crafted packet from a virtual machine (VM) to the vulnerable packet parser of the virtual switch. This results in taking control of the physical machine due to co-location of the virtual switch with the hypervisor. Next, she can take control of the hypervisor where the VM running the network—and in most cases cloud—controller is hosted due to the direct communication channel. From the controller, she can leverage the logically centralized design to, e.g., manipulate flow rules to violate essential network isolation.

Note that to successfully execute the above attack, it is sufficient for the attacker to rent a VM in the cloud. Besides increasing the attack surface which can be exploited with low resources, as our case study demonstrates, the impact of such an attack can be potentially much larger than usually considered in the context of software/hardware data planes [3, 4].

Contributions:

- We identify the adversary facing, and increasingly complex, unified packet parser as a new attack surface introduced by virtual switches.
- We introduce a relevant attacker model for virtual switches, namely, the virtual switch attacker model for packet-parsing (vAMP).
- Using a case study with OpenStack and OvS, we demonstrate that existing virtual switches cannot withstand the vAMP attack.

Ethical Considerations: To avoid disrupting the normal operation of businesses, we verified our findings on our own infrastructure. We have disclosed our findings to the OvS team who have integrated the fixes. Ubuntu, Redhat, Debian, Suse, Mirantis, and other stakeholders have applied these fixes in their stable releases. Furthermore, CVE-2016-2074 and CVE-2016-10377 were assigned to our discoveries.

2 UNIFIED PACKET PARSING: A NEW ATTACK SURFACE

The packet parser is typically integrated into the code base of the virtual switch. It is the first component of the virtual switch to process network packets it receives from the physical or virtual network interface. Therefore, it can receive, and process attacker controlled packets. The network protocols commonly parsed by virtual switches are from Layer 2 to Layer 4, i.e., from Ethernet up to TCP/UDP. Workload increase and advancements in network virtualization are driving virtual switches to implement middlebox functions such as load balancing, deep packet inspection, and stateful firewalls [5]. This necessitates two major requirements of virtual switches: (i) Support the parsing of more or new protocols; (ii) maintain if not improve existing forwarding performance.

Unified packet parsing addresses the above requirements. All the supported header fields of a packet are extracted in a single pass instead of multiple passes. This enables the virtual switch and other network functions to operate on packet

metadata instead of the actual packet. Packet operations are left to the end of the packet processing pipeline. Naturally, this approach performs better than parsing or re-parsing headers on a need-to basis [6].

Although this centralizes the parsing code, and has the potential to ease verification [7], it is also the most likely piece of code to contain security vulnerabilities [2]. Furthermore, a security vulnerability here, can compromise all the dependant mechanisms, and policies, e.g., firewalls and policies for network isolation among tenants. Thus, the attack surface of the virtual switch increases with any new protocol that is included in parsing. Recall from Fig. 1, that the number of supported protocols in virtual switch implementations such as OvS, and Cisco's Nexus 1000V are on the rise.

3 ATTACKER MODELS

Having identified the unified packet parser as potentially increasing the attack surface for virtual switches, we now revisit existing attacker models to investigate whether virtual switches are appropriately accounted for. In particular we look at prior work starting from 2009 which is the year when virtual switches emerged into the virtualization market [1]. Subsequently, motivated by the shortcomings of existing attacker models, we introduce a novel attacker model: the vAMP attack.

3.1 Prior Attacker Models for Virtual Switches

Virtual switches intersect with several areas of network security research: Data plane, network virtualization, software defined networking (SDN), and the cloud. Therefore, we conducted a qualitative analysis that includes research we identified as relevant to attacker models for virtual switches. In the following we elaborate on that.

A general attacker model for the data plane was proposed by Chasaki et al. [8] that involves the attacker exploiting the packet processor of routers. Keller et al. [4] proposed an attacker model for router virtualization that assumed the virtualization layer in general can be compromised. Qubes OS [9] also makes a general assumption that the networking stack can be compromised. Similarly, Dhawan et al. [10] assumed that the Software Defined Network (SDN) data plane can be compromised.

Paladi et al. [11] conservatively assumed the Dolev-Yao model for network virtualization in a multi-tenant cloud. Grobauer et al. [12] observed that virtual networking can be attacked in the cloud without a specific attacker model.

Jin et al. [3] accurately identified two threats faced by virtual switches: Virtual switches are co-located with the hypervisor; and guest VMs need to interact with the hypervisor. However, they stopped short of providing a concrete threat model, and underestimated the impact of compromising virtual switches. Indeed at the time, cloud systems were burgeoning. Only recently, Alhebaishi et al. [13] proposed an updated approach to cloud threat modelling. The virtual switch was identified as a component of cloud systems that needs to be protected. However, the severity, and multitude of threats that apply to virtual switches was overlooked.

Motivated by a strong adversary, Gonzales et al. [14], and Karmakar et al. [15] accounted for virtual switches, and the data plane. Similarly Yu et al. [16] assumed a strong adversarial model, with an emphasis on hardware switches, and the defender having sufficiently large resources.

Therefore, what we observe is that previous work have either provided a general attacker model for the data plane, fallen short of providing an accurate attacker model for virtual switches, underestimated the impact of a compromised virtual switch, or assumed strong attackers. We find this to be inadequate given the important role virtual switches play in the cloud, network (function) virtualization, and SDN. Hence, what is needed is an accurate, and relevant attacker model for virtual switches, which we describe next.

3.2 The Virtual Switch Attacker Model for Packet-Parsing (vAMP)

Given the shortcomings of the above attacker models, we now present a new attacker model for virtual switch based cloud network setups that use a logically centralized controller. Contrary to prior work we identify the virtual switch as a critical core component which has to be protected against direct attacks, e.g., malformed packets. Furthermore, our attacker is not supported by a major organization (she is a "Lone Wolf") nor does she have access to special network vantage points. The attacker's knowledge of computer programming and code analysis tools is comparable to that of an average software developer. In addition, the attacker controls a computer that can communicate with the cloud under attack.

The attacker's target is a cloud infrastructure that uses virtual switches for network virtualization. We assume that our attacker has only limited access to the cloud. Specifically, the attacker does not have physical access to any of the machines in the cloud. Regardless of the cloud delivery model and whether the cloud is public or not, we assume the attacker can either rent a single VM, or has already compromised a VM in the cloud, e.g., by exploiting a web-application vulnerability [17].

We assume that the cloud *provider* follows security best-practices [18]. Hence, at least three isolated networks (physical/virtual) dedicated towards management, tenants/guests, and external traffic exist. Furthermore, we assume that the same software stack is used across all servers in the cloud.

We consider our attacker as successful, if the attacker obtains full control of the cloud. This means that the attacker can perform arbitrary computation, create/store arbitrary data, and send/receive arbitrary data to all nodes including the Internet.

4 TAKING CONTROL OF THE CLOUD

Based on our analysis, we conjecture that current virtual switch implementations are not robust to adversaries from our attacker model. Indeed it is generally known that vulnerabilities exist, and will occur, even in well maintained production software. However, the point we make here is that finding vulnerabilities can be easily accomplished by amateurs, e.g., by fuzzing; moreover, a single vulnerability can have a devastating impact.

For the purpose of our case study, we evaluated the virtual switch Open vSwitch in the context of the cloud operating system OpenStack against our attacker model. We opted for this combination as OpenStack is one of the most prominent cloud systems, with thousands of production deployments in large enterprises and small companies alike. Furthermore, according to the OpenStack Survey 2016 [19], over 60% of OvS deployments are in production use and over one third of 1000+ core clouds surveyed use OvS.

Attack Methodology: We conducted a structured attack targeted at the attack surface we identified, i.e., the unified packet parser. We expected to find vulnerabilities in the unified packet parser code of OvS, which executes in the *ovs-vswitchd* process. Hence, our next step used an off-the-shelf coverage-guided fuzz tester, namely American Fuzzy Lop (AFL), on OvS's unified packet parser code. Specifically, for our tests we used AFL version 2.03b and source code of OvS version 2.3.2 recompiled with AFL instrumentation. Following common best practice for fuzzing code, all crashes reported by the fuzzer were triaged to ascertain their root cause.

We identified several vulnerabilities by fuzzing the unified packet parser of OvS, these include two stack buffer overflows, and one integer underflow. In this paper we focus on only one of the vulnerabilities we found as it suffices to demonstrate the attack. The vulnerability is a stack buffer overflow in the MPLS label stack parsing code of OvS.

The stack buffer overflow occurs when a large MPLS label stack packet is parsed beyond the defined threshold. As predicted, this attack has its root-cause in the unified packet parser. Indeed, we note that the specification of MPLS (see RFC 3031 [20] and RFC 3032 [21]) does not specify how to parse the whole label stack. Instead, it specifies that when a packet with a label stack arrives at a forwarding component, only the top label must be popped to make a forwarding decision. Yet, OvS parses all labels of the packet even beyond the

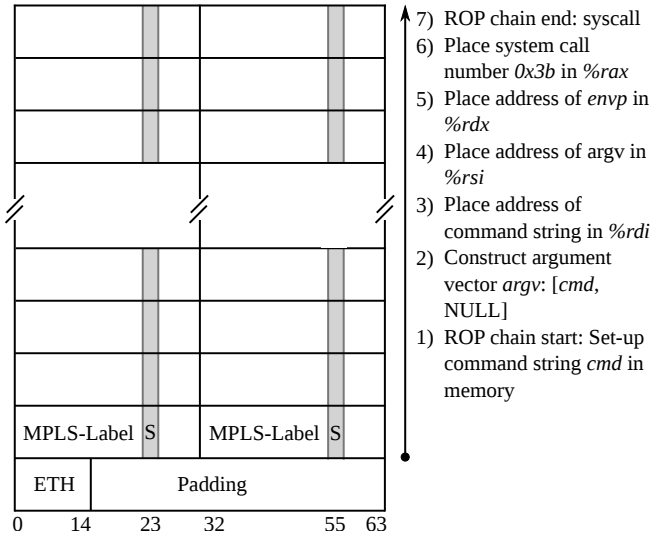

Figure 3: A visual representation of our ROP chain (in an Ethernet frame) for *ovs-vswitchd* to spawn a shell and redirect it to a remote socket address.

supported limit and beyond the pre-allocated memory range for that stack. If OvS would handle MPLS packets as per the specifications, it would only pop the top label, which has a static, defined size. Thus, there would be no opportunity for a buffer overflow.

The next step was to convert the vulnerability into an exploit that gives us a shell on the remote system. We crafted the exploit by creating a ROP [22] attack hidden in an MPLS packet as illustrated in Fig. 3. One of the challenges we faced in creating the exploit was the following. As per RFC 3032, MPLS label processing terminates if the *S* bit is set to 1. Therefore, to obtain a successful ROP chain, i.e., to prevent early termination of our exploit, we had to select appropriate gadgets by customizing Ropgadget [23] and modifying the shell command string accordingly. The constraint on the *S* bit for the gadgets in the MPLS labels is shown in Fig. 3 as the gray lines.

Figure 3 also depicts the ROP chain in our exploit packet, starting with the Ethernet header and padding, followed by the MPLS labels. Our example ROP payload connects a shell on the victim's system (the server running *ovs-vswitchd*) to a listening socket on the remote attacker's system. To spawn the shell the payload triggers the execution of the *cmd* `bash -c "bash -i >& /dev/tcp/<IP>/<PORT> 0>&1"` through the *execve* system call (0x3b). This requires the following steps: 1) Set-up the shell command (*cmd*) string in memory; 2) construct the argument vector *argv*; 3) place the address of the command string in the register *%rdi*; 4) place the address of *argv* in *%rsi*; 5) place the address of *envp* in *%rdx*; 6) place

the system call number $0x3b$ in $\%rax$; and finally 7) execute the system call, *execve*.

With a successful ROP chain, the next step was to create a worm. We need multiple steps to propagate the worm. These are visualized in Figure 2. In Step 1, the worm originates from an attacker-controlled (guest) VM within the cloud and compromises the host operating system (OS) of the server via the vulnerable packet processor of the virtual switch. Once she controls the server, she patches *ovs-vswitchd* on the compromised host, as otherwise the worm packet cannot be propagated. Instead the packet would trigger the vulnerability in OvS yet again.

With the server under her control the remote attacker, in Step 2, propagates the worm to the hypervisor running the controller VM and compromises it via the same vulnerability. The centralized architecture of OpenStack requires the controller to be reachable from all servers via the management network and/or guest network. By gaining access to one server we gain access to these networks and, thus, to the controller. Indeed, the co-location of the data plane and the controller, provides the necessary connectivity for the worm to propagate from any of the servers to the controller. Network isolation using VLANs and/or tunnels (GRE, VXLAN, etc.) does not prevent the worm from spreading once the server is compromised.

With the controller's server also under the control of the remote attacker, the worm again patches *ovs-vswitchd* and can then taint the remaining uncompromised server(s) (Step 3). Thus, finally, after Step 3, all servers are under the control of the remote attacker. We automated the above steps using a shell script. Our worm exploit could also have been created by an attacker with average programming skills who has some experience with this kind of technique. This is in accordance with our attacker model, which does not require an uncommonly skilled attacker.

Attack Evaluation: Rather than evaluating the attack in the wild we chose to create a test setup in a lab environment. More specifically, we used the Mirantis 8.0 distribution that ships OpenStack "*Liberty*" with OvS version 2.3.2. The test setup consisted of a server (the fuel master node) that can configure and deploy other OpenStack nodes (servers) including the OpenStack controller, compute, storage and network. Due to limited resources, we created one controller and one compute node with multiple VMs in addition to the fuel master node using the default Mirantis 8.0 configuration. Virtual switching was handled by OvS.

The attacker was given control of one of the VMs on the compute server and could deploy the worm from there. It took less than 20 seconds until the worm compromised the controller. This means that the attacker has root shell (ovs-vswitchd runs as root) access to the compute node as well as the controller. This includes 3 seconds of download time for patching *ovs-vswitchd* (OvS user-space daemon), the shell script, and the exploit payload. Moreover, we added 12 seconds of sleep time for restarting the patched *ovs-vswitchd* on the compute node so that attack packets could be forwarded.

Next, we added 60 seconds of sleep time to ensure that the network services on the compromised controller were restored. Since all compute nodes are accessible from the controller, we could compromise them in parallel. This takes less time than compromising the controller, i.e., less than 20 seconds. Hence, we conclude that the compromise of a standard cloud setup can be performed in less than two minutes.

Attack Result: Our evaluation demonstrates how easily an amateur attacker can compromise the virtual switch, and subsequently take control of the entire cloud in a matter of minutes. This can have serious consequences, e.g., amateur attackers can exploit virtual switches to launch ransomware attacks in the cloud. This is a result of complex packet parsing in the unified packet parser, co-locating the virtual switch with the virtualization layer, and incorrect attacker models.

5 CONCLUDING REMARKS

While this paper has focussed on virtual switches, the virtualization trend, and hence the relevance of our attacker model is more general. In particular, additional network functionality, e.g., middleboxes, is migrated from hardware equipment to commodity servers [5, 6], not only in the cloud but also other infrastructure providers, e.g., Internet Service Providers [24]. Furthermore, as we move towards realizing SDN 2.0 [25], network virtualization and higher layer protocols (Layer 4-7) will play pivotal roles. Hence, it is imperative for data plane elements to be more robust to attacks, especially in vulnerable code segments such as the unified packet parser.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their valuable feedback and comments. The authors would like to express their gratitude towards the German *Bundesamt für Sicherheit in der Informationstechnik*, for sparking the authors' interest in SDN security. This work was partially supported by the Helmholtz Research School in Security Technologies scholarship, Danish Villum Foundation project "ReNet", BMBF (Bundesministerium für Bildung und Forschung) Grant KIS1DSD032 (Project Enzevalos), and by the Leibniz Prize project funds of DFG/German Research Foundation (FKZ FE 570/4-1). We would also like to thank the security team at Open vSwitch for their timely response.

REFERENCES

[1] Ben Pfaff et al. "Extending Networking into the Virtualization Layer." In: *Proc. ACM Workshop on Hot Topics in Networks (HotNETs)*. 2009.

[2] Len Sassaman et al. "Security Applications of Formal Language Theory". In: *IEEE Systems Journal* 7.3 (Sept. 2013).

[3] Xin Jin, Eric Keller, and Jennifer Rexford. "Virtual Switching Without a Hypervisor for a More Secure Cloud". In: *Proc. USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (HotICE)*. 2012.

[4] Eric Keller, Ruby B. Lee, and Jennifer Rexford. "Accountability in Hosted Virtual Networks". In: *Proc. ACM Workshop on Virtualized Infrastructure Systems and Architectures*. VISA '09. 2009.

[5] Ethan J. Jackson et al. "Softflow: A middlebox architecture for open vswitch". In: *Usenix Annual Technical Conference (ATC)*. 2016.

[6] Daniel Firestone. "VFP: A Virtual Switch Platform for Host SDN in the Public Cloud." In: *Proc. Usenix Symposium on Networked Systems Design and Implementation (NSDI)*. 2017.

[7] Mihai Dobrescu and Katerina Argyraki. "Software Dataplane Verification". In: *Proc. Usenix Symposium on Networked Systems Design and Implementation (NSDI)*. Apr. 2014.

[8] Danai Chasaki and Tilman Wolf. "Attacks and Defenses in the Data Plane of Networks". In: *Proc. IEEE/IFIP Transactions on Dependable and Secure Computing (DSN)* 9.6 (Nov. 2012).

[9] Joanna Rutkowska and Rafal Wojtczuk. "Qubes OS architecture". In: *Invisible Things Lab Tech Rep* 54 (2010).

[10] Mohan Dhawan et al. "SPHINX: Detecting Security Attacks in Software-Defined Networks." In: *Proc. Internet Society Symposium on Network and Distributed System Security (NDSS)*. 2015.

[11] Nicolae Paladi and Christian Gehrmann. "Towards Secure Multi-tenant Virtualized Networks". In: *Proc. IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. Aug. 2015.

[12] Bernd Grobauer, Tobias Walloschek, and Elmar Stöcker. "Understanding Cloud Computing Vulnerabilities". In: *IEEE Security & Privacy Magazine* 9.2 (Mar. 2011).

[13] Nawaf Alhebaishi et al. "Threat Modeling for Cloud Data Center Infrastructures". In: *Intl. Symposium on Foundations and Practice of Security*. Springer. 2016.

[14] Dan Gonzales et al. "Cloud-Trust - a Security Assessment Model for Infrastructure as a Service (IaaS) Clouds". In: *Proc. IEEE Conference on Cloud Computing* PP.99 (2017).

[15] Kallol Krishna Karmakar, Vijay Varadharajan, and Uday Tupakula. "Mitigating attacks in Software Defined Network (SDN)". In: *Proc. IEEE Software Defined Systems (SDS)*. May 2017.

[16] Dongting Yu et al. *Security: a Killer App for SDN?* Tech. rep. Indiana Uni. at Bloomington, 2014.

[17] A. Costin. "All your cluster-grids are belong to us: Monitoring the (in)security of infrastructure monitoring systems". In: *Proc. IEEE Communications and Network Security (CNS)*. Sept. 2015.

[18] *OpenStack Security Guide*. http://docs.openstack.org/security-guide. Accessed: 27-01-2017. 2016.

[19] Heidi J. Tretheway et al. "A snapshot of Openstack users' attitudes and deployments." In: *Openstack User Survey* (Apr 2016).

[20] Eric C. Rosen, Arun Viswanathan, and Ross Callon. *Multiprotocol Label Switching Architecture*. RFC 3031 (Proposed Standard). Internet Engineering Task Force, Jan. 2001. URL: http://www.ietf.org/rfc/rfc3031.txt.

[21] Eric C. Rosen et al. *MPLS Label Stack Encoding*. RFC 3032 (Proposed Standard). Internet Engineering Task Force, Jan. 2001. URL: http://www.ietf.org/rfc/rfc3032.txt.

[22] Ryan Roemer et al. "Return-Oriented Programming: Systems, Languages, and Applications". In: *ACM Trans. on Information and System Security (TISSEC)* 15.1 (Mar. 2012).

[23] *ROPGadget Tool*. https://github.com/JonathanSalwan/ROPgadget/tree/master. Accessed: 02-06-2016.

[24] Sue Marek. *ATT's Stephens: 47% of Network Functions Are Virtualized*. https://www.sdxcentral.com/articles/news/atts-stephens-47-network-functions-virtualized/2017/07/. Accessed: 28-07-2017. 2017.

[25] Craig Matsumoto. *Time for an SDN Sequel? Scott Shenker Preaches SDN Version 2*. https://www.sdxcentral.com/articles/news/scott-shenker-preaches-revised-sdn-sdnv2/2014/10/. Accessed: 27-01-2017. 2014.