# Taking Control of SDN-based Cloud Systems via the Data Plane

Thimmaraju, Kashyap; Shastry, Bhargava; Fiebig, Tobias; Hetzelt, Felicitas; Seifert, Jean-Pierre; Feldmann, Anja; Schmid, Stefan

**DOI**
10.1145/3185467.3185468

**Publication date**
2018

**Document Version**
Accepted author manuscript

**Published in**
Proceedings of ACM Symposium on SDN Research (SOSR)

# Taking Control of SDN-based Cloud Systems via the Data Plane

Kashyap Thimmaraju
Security in Telecommunications
TU Berlin
Berlin, Germany
kash@sect.tu-berlin.de

Bhargava Shastry
Security in Telecommunications
TU Berlin
Berlin, Germany
bshastry@sect.tu-berlin.de

Tobias Fiebig
Faculty of Technology, Policy and Management
TU Delft
Delft, Netherlands
t.fiebig@tudelft.nl

Felicitas Hetzelt
Security in Telecommunications
TU Berlin
Berlin, Germany
file@sect.tu-berlin.de

Jean-Pierre Seifert
Security in Telecommunications
TU Berlin
Berlin, Germany
jpseifert@sect.tu-berlin.de

Anja Feldmann
Internet Architecture
Max-Planck-Institut für Informatik
Saarbrücken, Germany
anja@mpi-inf.mpg.de

Stefan Schmid*†
Faculty of Computer Science
University of Vienna
Vienna, Austria
schmiste@univie.ac.at

## ABSTRACT

Virtual switches are a crucial component of SDN-based cloud systems, enabling the interconnection of virtual machines in a flexible and "software-defined" manner. This paper raises the alarm on the security implications of virtual switches. In particular, we show that virtual switches not only *increase the attack surface* of the cloud, but virtual switch vulnerabilities can also lead to attacks of much *higher impact* compared to traditional switches.

We present a systematic security analysis and identify four design decisions which introduce vulnerabilities. Our findings motivate us to revisit existing threat models for SDN-based cloud setups, and introduce a new attacker model for SDN-based cloud systems using virtual switches.

We demonstrate the practical relevance of our analysis using a case study with Open vSwitch and OpenStack. Employing a fuzzing methodology, we find several exploitable vulnerabilities in Open vSwitch. Using just one vulnerability we were able to create a worm that can compromise hundreds of servers in a matter of minutes.

Our findings are applicable beyond virtual switches: NFV and high-performance fast path implementations face similar issues. This paper also studies various mitigation techniques and discusses how to redesign virtual switches for their integration.

## KEYWORDS

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

## 1 INTRODUCTION

Modern cloud systems such as OpenStack [7], Microsoft Azure [26] and Google Cloud Platform [92] are designed for programmability, (logically) centralized network control and global visibility. These tenets also lie at the heart of Software-defined Networking (SDN) [23, 51] which enables cloud providers to efficiently utilize their resources [35], manage their multi-tenant networks [44], and reason about orchestration [41].

The data plane of Software-Defined Networks in the cloud are highly virtualized [44]: Virtual switches (running on the servers) are responsible for providing connectivity and

*Also with, Internet Network Architectures, TU Berlin.
†Also with, Dept. of Computer Science, Aalborg University.


*SOSR '18, March 28–29, 2018, Los Angeles, CA, USA*

ACM ISBN 978-1-4503-5664-0/18/03…$15.00
https://doi.org/10.1145/3185467.3185468

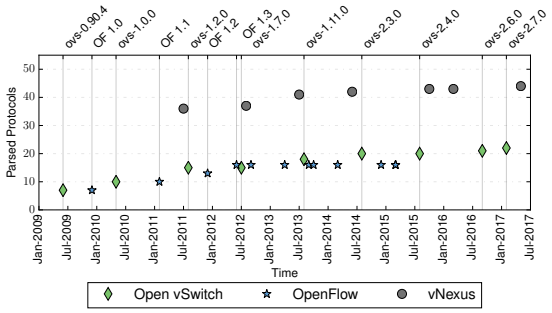



**Figure 1: The total number of parsed high-level protocols in two popular virtual switches and OpenFlow from 2009-2017.**

*isolation* among virtual machines [63]. Prominent virtual switches today are: Open vSwitch (OvS) [64], Cisco Nexus 1000V [93], VMware vSwitch [94] and Microsoft VFP [26].

Virtual switches are typically not limited to provide *traditional switching* but support an increasing number of network and middlebox functionality [26, 33], e.g., routing, firewalling, network address translation and load-balancing. Placing such functionality at the virtualized edge of the network (i.e., the servers) is attractive, as it allows to keep the network fabric simple and as it supports scalability [26, 63].

However, the trend to move functionality from the network fabric to the edge (virtual switch) also comes at the price of increased complexity. For example, the number of protocols that need to be parsed and supported by virtual switches (Open vSwitch and Cisco Nexus 1000v) and OpenFlow [51] have been growing steadily over the last years [89] (see Fig. 1).

The trend towards more complex virtual switches is worrisome as it may increase the attack surface of the virtual switch. For example, implementing network protocol parsers in the virtual switch is non-trivial and error-prone [25, 79, 82]. These observations lead us in this paper to conduct a security study of virtual switches.

**Our contributions:**

- We present a systematic security analysis of virtual switches. We find that virtual switches not only increase the attack surface of an SDN-based cloud system (compared to their traditional counterparts), but can also have a much larger impact on cloud systems.
- Our analysis reveals four main factors that cause security issues: The co-location of virtual switches with the server's virtualization layer (in user- and kernel-space); centralized control; complex packet parsing (and processing) of attacker controlled data.
- Our findings motivate us to revisit current threat models. We observe that existing models do not encompass the security issues identified in this paper leading us to introduce a new attacker model for the operation of virtualized data plane components in a Software-defined Network as well as in the context of Network Function Virtualization (NFV): A *low-budget attacker* can cause *significant* harm on SDN-based cloud systems.
- We demonstrate the *practical* feasibility of our attacks on OvS, a popular open-source virtual switch implementation used in SDN-based cloud systems. This case study shows that commonly used virtual switch implementations are not resilient against our attacker model. Indeed, such an attacker can successfully exploit a whole SDN-based cloud setup within minutes.
- We extend our study by surveying high performance fast paths, other virtual switch implementations, and related SDN and NFV technologies. We find that they are also susceptible to the same design issues. Furthermore, we find that software mitigations are commonly not considered during the evaluation of new data plane components.
- We find that software mitigations for the vulnerabilities we exploited could be adopted with a small performance penalty for real-world traffic scenarios. Their use must be evaluated during design and implementation of new SDN and NFV components.

**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 the discovered vulnerabilities.

**Structure:** We provide necessary background information on virtual switches in Section 2. Section 3 introduces and discusses our security analysis of virtual switches and existing threat models. Based on this analysis we propose a new attacker model. Section 4 presents a proof-of-concept case study attack on OvS in OpenStack. We then investigate how our findings on OvS relate to other virtual switches, high performance fast paths and SDN/NFV in Section 5. Subsequently, we discuss possible software mitigations and their performance impact in Section 6, and design countermeasures in Section 7. After discussing related work in Section 8, we conclude in Section 9.

## 2 BACKGROUND

This section reviews the background necessary to understand the remainder of this paper.

## 2.1 Virtual Switches

The network's data plane(s) can either be distributed across virtualized servers or across physical (hardware) switches. OvS, VMware vSwitch, and Cisco Nexus 1000V are examples of the former and are commonly referred to as *virtual switches*. Cisco VN-Link [2] and Virtual Ethernet Port Aggregator (VEPA) [38] are examples of the latter.

A virtual switch has two main components: control and data plane. The control plane handles management and configuration, i.e., the administration of the virtual switch (e.g., configuring ports, policies, etc.). The data plane is responsible for forwarding. This functionality can be spread across the system running the virtual switch. The virtual switch can, but does not have to, be separate processes. Moreover, it can either fully reside in user- or kernel-space, or be split across them.

Forwarding is usually based on a sequential (or circular) packet processing pipeline. The pipeline starts by parsing the packet's header to extract the information that is required for a lookup of the forwarding instructions for that packet. The lookup is typically a (flow) table lookup—the second stage of the pipeline. The final stage uses this result to either forward the packet, drop it, or send it back to the first stage.

## 2.2 Open vSwitch

Open vSwitch (OvS) [14, 63, 64, 88] is a popular open source SDN and multi-platform virtual switch. OvS uses two forwarding paths: the slow path—a user-space daemon (*ovs-vswitchd*) and the fast path—a datapath kernel module (*openvswitch.ko*). *ovs-vswitchd* installs rules and associated actions on how to handle packets in the fast path, e.g., forward packets to ports or tunnels, modify packet headers, sample packets, drop packets, etc. When a packet does not match a rule of the fast path, the packet is sent to *ovs-vswitchd*, which then determines, in user-space, how to handle the packet. It then passes the packet back to the datapath kernel module to execute the action.

To improve performance for future packets, flow caching is used. OvS supports two main flavors of flow caching: *microflow caching* and *megaflow caching*. Oversimplifying things slightly, the former builds rules for individual connections, while the latter relies on generalization: It automatically determines the most general rule for handling a set of microflows. The latter can significantly reduce the number of required rules in the fast path.

## 2.3 MPLS

As our case study takes advantage of the MPLS (MultiProtocol Label Switching) parser, we include a brief overview here. MPLS is often deployed to address the complexity of per packet forwarding lookups, traffic engineering, and advanced path control. MPLS uses "Forwarding Equivalence Classes" (FECs) to place a "label" in the *shim header* between the Ethernet and the IP header [76] of a packet. This label is then used for forwarding. In addition, labels can be stacked via *push* and *pop* operations.

An MPLS label is 20 bits long, followed by the Exp field of 3 bits reserved space. This is followed by the 1 bit S field, which, if set to 1, indicates that the label is the bottom of the label stack. It is a critical piece of "control" information that determines how an MPLS node parses a packet. The TTL field indicates the Time-To-Live of the label.

MPLS labels should be under the providers' administration, e.g., offering L2/L3 VPNs, and are negotiated using protocols such as LDP (Label Distribution Protocol) [10], As per RFC 3032, MPLS labels are inherently trusted.

# 3 SECURITY ANALYSIS

In this section, we present a systematic security analysis of virtual switches. Based on these insights, we first investigate existing threat models for virtual switches and then construct an attacker model against which virtual switches must be resilient.

## 3.1 Attack Surface and Vulnerabilities

In the following we characterize the attack surface and vulnerabilities of virtual switches which make them feasible, attractive, and exploitable targets. An overview of the security analysis and the implications is illustrated in Fig. 2.

**Hypervisor co-location:** The design of virtual switches co-locates them—in SDN cloud setups—with the Host system and at least partially with the Host's kernel, see Figure 2. Components of the virtual switch slow-path often run with elevated (root) privileges in user-space on the Host system. From a performance perspective this is a sensible choice. However, from a security perspective this *co-location* and *elevated privilege* puts all virtual machines of the hypervisor at risk once an attack against the virtual switch is successful. Recall, such VMs include those that run critical cloud software, e.g., the VM hosting the controller.

**Centralized control via direct communication:** In an SDN the controller is tasked with all control plane decisions for every data plane component. Hereby, the controller uses its "southbound interface", today most often "OpenFlow", to communicate with all data plane elements—here the virtual switches. In a data center following industry best practises [6] this is often implemented using a trusted management network that is shared by all the data plane elements. This implies that a compromised data plane component can *directly* send packets towards the *controller* and/or *all* other *data plane elements*. Management networks, containing only

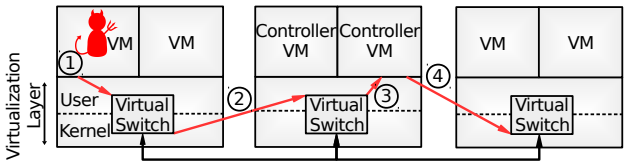



**Figure 2: An overview of the security implications of current virtual switch designs.**

trusted components, are commonly not protected with an additional intrusion detection system.

**Unified packet parser:** Once a virtual switch receives a packet it parses its headers to determine if it already has a matching flow rule. If this is not the case it will forward the packet to an intermediate data path (slow path) that processes the packet further in order to request a new flow table entry. In this step, the virtual switch commonly extracts all header information from the packet, e.g., MPLS and application layer information, before requesting a flow table entry from the controller. Parsing is the switch's responsibility as centralizing this task would not scale. The additional information from higher-level protocols is needed for advanced functionality like load balancing, deep packet inspection (DPI), and non-standard forwarding (see Section 5 for an overview of related technologies using these features in their implementation). However, with *protocol parsing in the data plane* the virtual switch is as susceptible to security vulnerabilities as any daemon for the parsed protocol. Thus, the attack surface of the data plane increases with any new protocol that is included in parsing.

**Untrusted input:** Virtual switches are commonly deployed in data centers at the network edge. This implies that virtual switches receive network packets directly from the virtual machines, typically unfiltered, see Section 2. This can be abused by an attacker. She can—via a virtual machine—send arbitrary data to a virtual switch[1]. Indeed, the virtual switch is typically the *first* data plane component to handle any packet from a VM. This enables attackers to take advantage of data plane vulnerabilities in virtual switches.

**Summary:** In combination, the above observations demonstrate why data plane attacks are a *feasible* threat and how they can spread throughout a cloud setup, see Fig. 2. By renting a VM and weaponizing a protocol parsing vulnerability an attacker can start her attack by taking over a single virtual switch (Step 1). Thus, she also takes control of the physical machine on which the virtual switch is running due to hypervisor co-location. Next (Step 2), she can take control of the Host OS where the VM running the network—and in most cases cloud—controller is hosted due to the direct communication channel. From the controller (Step 3), the attacker can leverage the logically centralized design to, e.g., manipulate flow rules to violate essential network security policies (Step 4). Alternatively, the attacker can change other cloud resources, e.g., modify the identity management service or change a boot image for VMs to contain a backdoor.

[1]Depending on the implementation, the *Dom0* IP stack may ensure that the IP part of all packets are well-formed.

## 3.2 Attacker Models for Virtual Switches

With these vulnerabilities and attack surfaces in mind, we revisit existing threat models. We particularly focus on work starting from 2009 when virtual switches emerged into the virtualization market [63]. We find that virtual switches are not appropriately accounted for in existing threat models, which motivates us to subsequently introduce a new attacker model.

**Existing threat models:** 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 cloud. In the following we elaborate on that.

Qubes OS [78] in general assumes that the networking stack can be compromised. Similarly, Dhawan et al. [20] assumed that the Software Defined Network (SDN) data plane can be compromised. Jero et al. [36] base their assumption on a malicious data plane in an SDN on Pickett's BlackHat briefing [65] on compromising an SDN hardware switch.

A conservative attacker model was assumed by Paladi et al. [55] who employ the Dolev-Yao model for network virtualization in a multi-tenant cloud. Grobauer et al. [28] observed that virtual networking can be attacked in the cloud without a specific attacker model.

Jin et al. [37] accurately described two threats to 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. However, only recently Alhebaishi et al. [9] proposed an updated approach to cloud threat modelling wherein the virtual switch was identified as a component of cloud systems that needs to be protected. However, in both cases, the authors overlooked the severity, and multitude of threats that apply to virtual switches.

Motivated by a strong adversary, Gonzales et al. [22], and Karmakar et al. [40] accounted for virtual switches, and the data plane. Similarly Yu et al. [97], Thimmaraju et al. [90] and Feldmann et al. [24] assumed a strong adversarial model,

with an emphasis on hardware switches, and the defender having sufficiently large resources.

Hence, we posit that previous work have either assumed a generic adversary model for the SDN data plane, stopped short of an accurate model for virtual switches, undervalued the impact of exploiting virtual switches, or assumed strong adversaries. Given the importance and position of virtual switches in general, and in SDN-based clouds in particular, we describe an accurate, and suitable attacker model for virtual switches in the following.

**A new attacker model:** 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 [6]. 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 successful, if she 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 CASE STUDY: OVS IN OPENSTACK

Based on our analysis, we conjecture that current virtual switch implementations are not robust to adversaries from our attacker model. In order to test our hypothesis, we conducted a case study. We evaluate 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 [91], over 60% of OvS deployments are in production use and over one third of 1000+ surveyed core clouds use OvS.

## 4.1 Attack Methodology

We conduct a structured attack targeted at the attack surface identified in our analysis.

**1. Attack surface analysis:** The first step of our analysis is validating co-location assumptions of OvS. We find that by default OvS is co-located with *Dom0's* user- and kernel-space, see Figure 2. Furthermore, the OvS daemon (*ovs-vswitchd*) has root privileges. Second, OvS supports logically centralized control and OpenFlow. See Section 2.2 for a more in-depth discussion of OvS. Finally, OvS implements a unified packet parser in its *key_extract* and *flow_extract* functions in the fast-past and slow-path resp.

**2. Vulnerability identification:** Based on our security analysis, we expect to find vulnerabilities in the unified packet parser of OvS. Hence, we used an off-the-shelf coverage-guided fuzz tester, namely American Fuzzy Lop (AFL), on OvS's unified packet parser in the slow-path. Specifically, for our tests we used AFL version 2.03b, source code of OvS version 2.3.2 recompiled with AFL instrumentation and the *test-flows* test case[81]. Following common best practice for fuzzing code, all crashes reported by the fuzzer were triaged to ascertain their root cause.

**3. Large-scale compromise:** The pure presence of a vulnerability is not sufficient to state that OvS is not robust against our threat model. We have to demonstrate that the vulnerability does enable a large-scale compromise. Thus, we need to turn the vulnerability into an exploit. Here, we use a common exploit technique, namely Return Oriented Programming (ROP) [75], to realize a worm that can fully compromise an OpenStack setup within minutes.

## 4.2 Identified Vulnerabilities

Using the above methodology, we identify several vulnerabilities in the unified packet parser of OvS (*ovs-vswitchd*). In this paper we only focus on one of the vulnerabilities we found in the stable branch (v2.3.2), as it suffices to demonstrate the attack. Further vulnerabilities discovered during our study include exploitable parsing errors leading to denial of service (DoS) (CVE-2016-2074) and an ACL bypass vulnerability (CVE-2016-10377) in the packet filter component of OvS.

The vulnerability is a stack buffer overflow in the MPLS parsing code of the OvS slow-path. We acknowledge that stack buffer overflows and how they are exploited are well

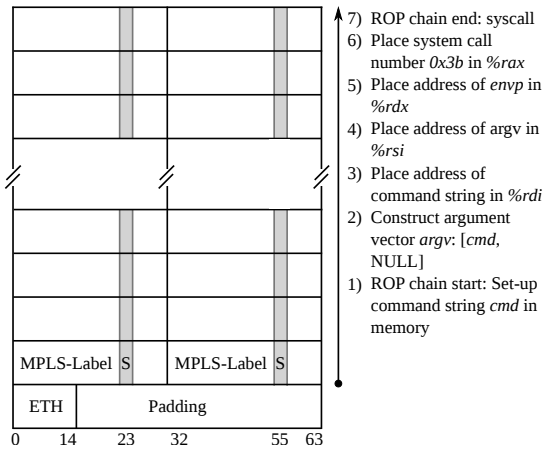



**Figure 3: A visual representation of our ROP chain (in an Ethernet frame) for a 64-bit version of *ovs-vswitchd* to spawn a shell and redirect it to a remote socket address. The gray columns indicate the position of the "S" bit in the MPLS label.**

understood. However, we fully document it here to: (i) Underline how easily such vulnerabilities can occur, especially in software handling network packets, and, (ii) To make our work more accessible in the context of networking research outside the security community.

The stack buffer overflow occurs when a large MPLS label stack packet that exceeds a pre-defined threshold is parsed. As predicted, this attack has its root-cause in the unified packet parser for MPLS. Indeed, we note that the specification of MPLS, see RFC 3031 [77] and RFC 3032 [76] 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 be used to make a forwarding decision. Yet, OvS parses all labels of the packet even beyond the supported limit and beyond the pre-allocated memory range for that stack. If MPLS would be handled correctly by OvS, it would only pop the top label, which has a static, defined size. Thus, there would be no opportunity for a buffer overflow.

## 4.3 Exploiting the Vulnerability as a Worm

Following our methodology, the next step is to show how the discovered vulnerability can be used by an attacker to compromise a cloud deployment. We start using the vulnerability to enable code execution on the virtual switch's host. Subsequently, we extend this to create a worm.

**Exploit:** The next step towards a full compromise is a remote-code-execution exploit based on the discovered vulnerability. We implement this by creating a ROP [75] attack hidden in an MPLS packet. By now, ROP attacks are well documented and can be created by an attacker who has explored the literature on implementing ROP attacks, e.g., using RopGadget [1]. Hence, we do not describe ROP here and suggest the reader to refer to Roemer et al. [75].

Recall from Sec. 2.3 that the MPLS label processing terminates if the *S* bit is set to 1. Therefore, to obtain a successful ROP chain, we select appropriate gadgets by customizing Ropgadget and modify the shell command string. 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*.

In summary, our 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.

**Worm Implementation:** We need multiple steps to propagate the worm. These are visualized in Figure 4. 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 server 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.

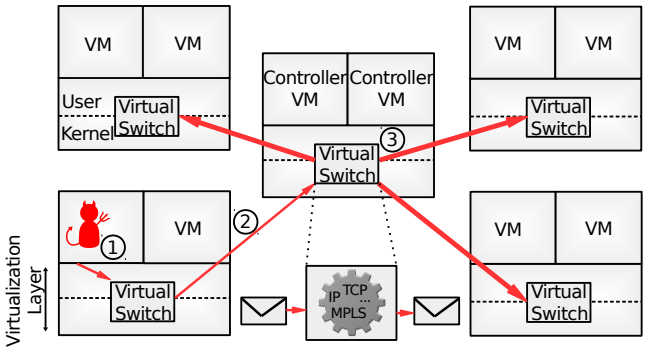



**Figure 4: In a typical cloud system, a worm can propagate to all the systems by exploiting security weaknesses of virtual switches: co-location, centralized and directed communication channels, and the unified packet parser.**

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.

## 4.4 Attack Evaluation

Rather than evaluating the attack in the wild we chose to create a test setup in a lab environment. More specifically, we use the Mirantis 8.0 distribution that ships OpenStack "*Liberty*" with OvS version 2.3.2. On this platform we set up multiple VMs. The test setup consists of a server (the fuel master node) that can configure and deploy other OpenStack nodes (servers) including the OpenStack controller, compute, storage, 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.

## 4.5 Summary

Our case study 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, centralized and direct control, and inadequate attacker models.

# 5 DISCUSSION: ANALYSIS OF RELATED TECHNOLOGIES

While so far we were mainly concerned with virtual switches (and in particular OvS in our case study), we believe that our work has ramifications far beyond. Our general observations apply not only to virtual switches across the board, but also to emerging NFV implementations and high-performance fast path implementations. Hence, in this section we evaluate, which other implementations and data-plane component classes are affected by our analysis. See Table 1 for a summary of our observations for some representative examples from each group.

**High Performance Fast Paths:** High performance fast paths (HPFPs) are software libraries for handling packet forwarding in user-space. Prominent examples include Data Plane Development Kit (DPDK) [32, 66] and NetMAP [72]. HPFPs try to minimize the performance bottlenecks of packet forwarding in the kernel. They accomplish this, by, e.g., using large page sizes, dedicated ring buffers, uniform packet format sizes, and improved buffer management. Thus, HPFPs can be used to increase forwarding performance in user-space virtual switches by eliminating the kernel (fast-path), e.g., OvS with DPDK [74].

Besides increasing virtual switch performance, an HPFP also increases security as it reduces packet processing in the kernel. This reduces the attack surface but does not fully address the problem of co-location since it is still running on the same host OS as the hypervisor. Moreover, we find that some HPFPs are not designed with software security in mind. Only IX [12] and Arrakis [59] are designed with the goal of improving packet handling security. NetMAP [72] at least discusses that not using shared memory with the host's kernel improves security. Furthermore, software mitigations

**Table 1: Attack surface summary for HPFPs, virtual switches, and SDN/NFV example implementations.**

| | Name | Ref. | Year | OvS based | Co-Location | Ext. Parsing | IOMMU | Soft. Mitigations | Sec. Focused | Comments |
|---|---|---|---|---|---|---|---|---|---|---|
| HPFP | DPDK | [67] | 2011 | | | | | | | |
| | NetMAP | [72] | 2012 | | | | | | | |
| | Arrakis | [59] | 2014 | | | | ✓ | | ✓ | |
| | IX | [12] | 2014 | | | | | | ✓ | |
| | ESWITCH | [53] | 2016 | | | | | | | |
| Virtual Switches | OvS | [62] | 2009 | | ◐ | ◐ | | | | Baseline |
| | Cisco NexusV | [93] | 2009 | | ◐ | ◐ | | ? | | Commercial |
| | VMware vSwitch | [94] | 2009 | | ◐ | ◐ | | ? | | Commercial |
| | Vale | [73] | 2012 | | ◐ | ◐ | | | | Using HPFP to increase perfomance. |
| | Hyper-Switch | [68] | 2013 | ✓ | ◐ | ◐ | | | | |
| | MS HyperV-Switch | [52] | 2013 | | ◐ | ◐ | | ? | | Commercial |
| | MS VFP | [26] | 2017 | | ◐ | ● | | ? | | Commercial |
| | NetVM | [31] | 2014 | | ◐ | ◐ | | | | Using HPFP to increase performance. |
| | Lagopus | [54] | 2014 | | ◐ | ◐ | | | | Different vSwitch with a featureset similar to OvS. |
| | fd.io | [95] | 2015 | | ◐ | ◐ | | | | Uses Vector Packet Processing, e.g., see Choi et al. [16]. |
| | mSwitch | [29] | 2015 | | ◐ | ◐ | | | | Using HPFP to increase performance. |
| | BESS | [13] | 2015 | | ◐ | ○ | | | | Similar to the Click modular router [43]. |
| | PISCES | [80] | 2016 | ✓ | ◐ | ◐ | | | | Uses a domain specific language to customize parsing. |
| SDN/NFV | Unify | [84] | 2014 | ✓ | ◐ | ● | | | | NFV Chaining |
| | ClickOS | [48] | 2014 | | ◐ | ● | | | | Places a software switch on virtualization host. |
| | EDEN | [11] | 2015 | | ◐ | ● | | | | Places EDEN on end-hosts; Parses more to enable NF. |
| | OVN | [61] | 2015 | ✓ | ● | ◐ | | | | Co-locates SDN controller with the hypervisor. |
| | SoftFlow | [33] | 2016 | ✓ | ◐ | ● | | | | Integrating middlebox functions in OvS; more parsing. |

Suspectibility to parameter: ○: less than OvS; ◐: similar to OvS; ●: more than OvS; ?: unknown;

to limit the impact of vulnerabilities are not used by either of them.

**Virtual Switch Implementations:** Our comparison of virtual switches in Table 1 uses OvS as the baseline. Competing commercial virtual switch products include Cisco's Nexus 1000V [93], the VMware vNetwork [94], Microsoft Hyper-V vSwitch [52] and Microsoft VFP [26]. These implementations suffer from the same *conceptual issues* that we identified in our attack surface and verified with OvS due to hypervisor co-location [60, 64]. Since they are closed-source software systems, we do not know specifics about their use of software mitigations. Notably, Microsoft VFP introduces middlebox functionality into their virtual switch thereby increasing the susceptibility due to parsing. Lagopus, another open-source virtual switch implementation lacks the same popularity as OvS, yet retains its design shortcomings [54].

Research projects in the area of virtual switches, e.g., Vale [73] and NetVM [31], are mainly focused on performance. Thus, they often rely on HPFPs. This decreases their co-location attack surface in comparison to plain OvS. However, since they commonly still use kernel modules and/or user mode components with elevated privileges, the principle attack vector is still there. Thus, using HPFPs does not have a significant impact on the security of such designs. Furthermore, to support, e.g., OpenFlow, they have to implement extended parsers for packet content. In contrast to the above projects we find that PISCES [80] reduces the attack surface by restricting the parser to the relevant part of the packet. Yet, its design focus on flexibility and extensibility increases the attack surface again. Similarly, fd.io uses Vector Packet Processing, e.g., see Choi et al. [16], to handle packets, e.g., in between containers, but also as an interface to conventional data-plane components. Yet, again, this packet processing and parsing component lacks security considerations and remains co-located with critical host components. Overall, we find that academic virtual switch proposals rarely focus on security or evaluate software mitigations for their virtual switch designs.

**Network Function Virtualization:** Network Function Virtualization (NFV) is a relatively new trend, whereby data plane network functions such as routers, firewalls, load balancers, intrusion detection systems, and VPN tunnel endpoints are moved from specialized devices to VMs. With SDNv2 [49], NFVs get folded into SDN via Virtualized Network Functions (VNFs). Here, VNFs are network function implementations that commonly use a virtual switch and add their functionality on top, decoupled from the underlying hardware. In principle, network functions need more complex parsing and processing. Hence, their attack surface is larger. Moreover, we find, that some NFV/VNF frameworks

are built on top of OvS as their virtual switch component. Thus, they suffer from the same attack vectors as OvS. Some proposals, e.g., such as EDEN [11], go a step further and suggest to move network functions to all end-hosts. Therefore, such proposals increase the attack surface by increasing the number of possibly affected systems. Moreover, none of the NFV solutions included in Table 1 consider software mitigations or have their focus on security.

For SDN, virtual switches are again central components. Moreover, we note that most current proposals of the SDN community, e.g., Open Virtual Network (OVN) [61], suggest to co-locate the SDN controller with the virtualization layer and data plane components. Thus, SDN is highly susceptible to the attack surface pointed out in this paper. With recursively virtualized SDNs [19] this attack surface will be increased even further.

**Summary:** Emerging technologies for improving performance of user-space fast-path packet processing slightly reduce the attack surface pointed out in this paper. However, contemporary virtual switches not employing HPFPs suffer from the same problems as we demonstrated in OvS. The root-cause lies in the shared architecture of such virtual switches that co-locates them (partially) with the Host system. In addition, new technologies like NFV are also affected. Similar to OvS, these technologies are commonly implemented across user- and kernel-space. In addition, these technologies heavily rely on parsing, e.g., in case of DPI and load balancing. Proposals such as EDEN even consider implementing such NFV components on *all* end-hosts, spreading the attack surface further. Finally, we find that software mitigations are typically not evaluated when designing data plane components, as the main focus is on performance rather than security.

## 6 SOFTWARE COUNTERMEASURES

There exist many mitigations for attacks based e.g., on buffer overflows, including MemGuard [18], control flow integrity [8], position independent executables (PIEs) [57], and Safe (shadow) Stack [46]. Any one of these severely reduces the impact of crucial, frequently occurring vulnerabilities like the one used as an example in this paper. However, due to their assumed performance overhead, especially on latency, they are commonly not deployed for virtualized network components.

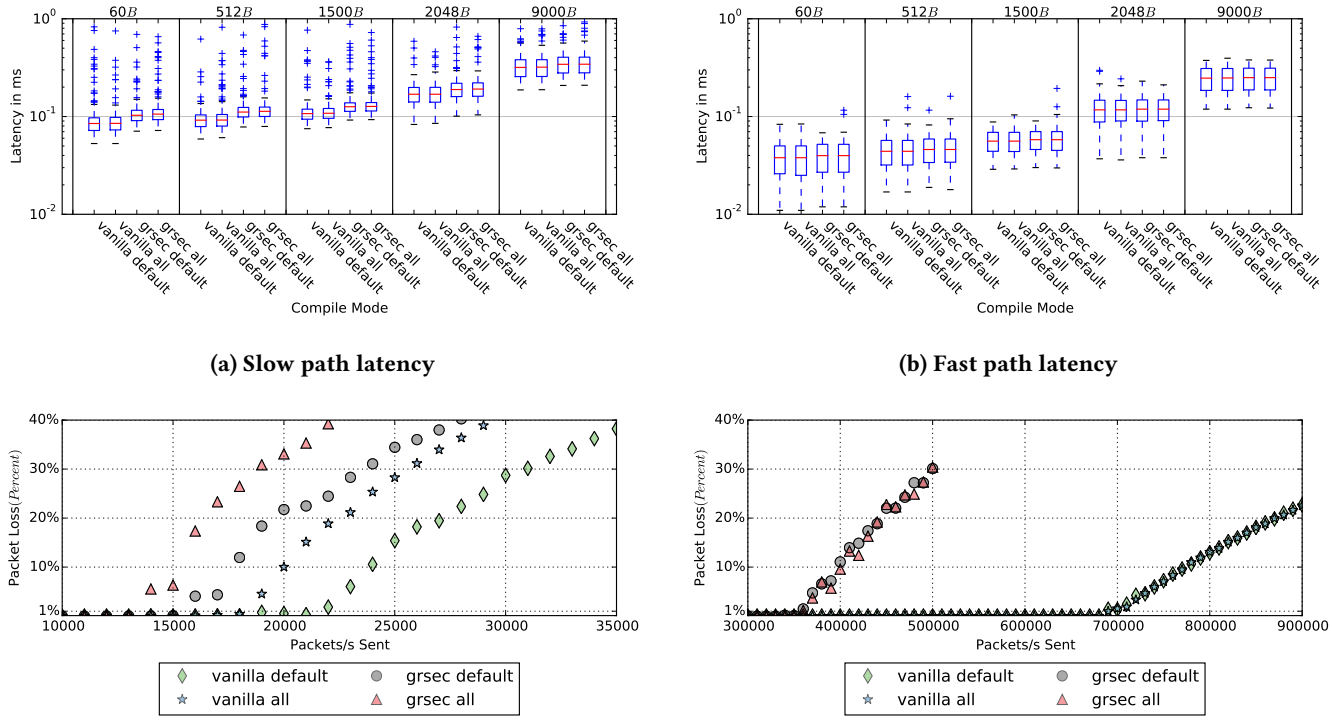
Hence, while these mitigations are widely available, we find that they are not enabled by default for OvS. Furthermore, virtual switch solutions presented in the literature commonly do not discuss these techniques. One possible downside of these mitigations is their performance overhead. Past work reported that MemGuard imposes a performance overhead of 3.5–10% [18] while PIEs have a performance impact of 3–26% [57]. Furthermore, prior evaluations did not focus on the systems' network performance. Instead, their main focus was on the systems' process performance, e.g., kernel context switches and the size of compiled binaries with the applied mitigations. However, in the context of OvS, network related metrics are far more relevant: Forwarding latency and forwarding throughput.

In order to investigate the potential performance penalty of such countermeasures, we showcase two variants of these mitigation techniques that are supported by the Gnu cc compiler `gcc` out of the box. Namely, stack protector and position independent executables. To determine the practical impact of these mitigations, we designed a set of experiments to evaluate the performance impact on OvS's forwarding latency and throughput.

**Evaluation Setup:** The test setup is chosen to ensure accurate one-way delay measurements. Thus, for our tests, we use three systems, all running Linux kernel (v4.6.5) compiled with gcc (v4.8). The systems have 16GB RAM, two dual-core AMD x86_64 2.5GHz, and four Intel Gigabit NICs. The systems are interconnected as follows: One system serves as the Load Generator (LG) and replays packet traces according to the specific experiments using *tcpreplay*. This system is connected to the Device Under Test (DUT), configured according to the different evaluation parameters. The data is then forwarded by OvS on the DUT to a Load Receiver (LR), a third system.

The connections between LG and DUT, and, LR and DUT respectively are monitored via a passive taping device. Both taps are connected to our measurement system. This system has two dual-core Intel(R) Xeon(TM) CPUs running at 3.73GHz with hyperthreading enabled and 16GB RAM. We use an ENDACE DAG 10X4-P card to capture data. Each line (RX/TX) of the tapped connections is connected to one interface of the DAG 10X4-P. Each interface has its own receive queue with 1GB. This ensures accurate one-way delay measurements with a high precision, regardless of the utilization of the measurement host.

**Evaluation Parameters:** We evaluate forwarding latency and throughput for eight different combinations of traffic composition and software mitigations. We compare a vanilla Linux kernel (v4.6.5) with the same kernel integrated with *grsecurity* patches (v3.1), which protects the in-kernel fast-path by preventing kernel stack overflow attacks using stack canaries, address space layout randomization and ROP defense. For both kernels, we evaluate two versions of OvS-2.3.2: The first one compiled with `-fstack-protector-all` for unconditional stack canaries and `-fPIE` for position independent executables; the second one compiled without these two features. Since *gcc*, the default compiler for the Linux kernel, does not support Safestack (safe and unsafe stack) we did not evaluate this feature, even though it will be available




(a) Slow path latency

(b) Fast path latency

(c) Slow path throughput

(d) Fast path throughput

**Figure 5: Forwarding performance of OvS, with and without countermeasures on a vanilla kernel and a grsecurity enabled kernel exclusively in the slow and fast path.**

with *clang*, another compiler, starting with version 3.8. The selected mitigations increase the total size of *ovs-vswitchd* from 1.84 MB to 2.09 MB (+13.59%) and *openvswitch.ko* from 0.16 MB to 0.21 MB (+31.25%). However, apart from embedded systems, the size changes are not relevant on modern systems with several hundred gigabytes of memory.

One important feature in virtual switches, recall Section 2, is, whether traffic is handled by the slow or the fast path. We decided to focus on the corner cases where traffic is either handled exclusively by the fast or by the slow path. By isolating the two cases we can assess if and to what extent the software security options impact each path. Hereby, we follow current best practices for OvS benchmarking, see Pfaff et al. [64]. To trigger the slow path for all packets in our experiments, we disable the *megaflows cache* and replay a packet trace in which each packet has a new source MAC address (via sequential increments). For measuring fast path performance, we pre-establish a single flow rule on the DUT, a wildcard-one, that matches all packets entering from the LG. The rule instructs the virtual switch to process these packets via the fast path and forward them on the interface connected to the LR. Therefore, for the sake of consistency, we can replay the same traces as used for the slow path experiments. Additionally, to reduce the uncertainty in our setup, we pin *ovs-vswitchd* to a single core.

**Latency Evaluation:** For the latency evaluation, we studied the impact of packet size on OvS forwarding. We selected the following packet sizes from the legacy MTU range: 60*B* (minimum IPv4 UDP packet size), 512*B* (average packet), and 1500*B* (maximum MTU) packets. In addition, we also select the following jumbo frames: 2048*B* packets (small jumbo frame) and 9000*B* (maximum jumbo frame). For each experimental run, i.e., packet size and parameter set, we continuously send 10,500 packets from the LG to the LR via the DUT at a rate of 10 packets per seconds (pps). To eliminate possible build-up or pre-caching effects, we only evaluate the last 10,000 packets of each experiment.

The results for the latency evaluation are depicted in Figures 5a and 5b for the slow path and fast path resp. We find that grsecurity (grsec default and grsec all) imposes a minimal increase in latency for all packet sizes in the slow and fast path. We observe a minimal impact of user-land protection mechanisms, 1-5%, see Figure 5a, for slow path latency, both, for a vanilla and a grsecurity enabled kernel. Naturally,

there is no impact of the user-land protection mechanisms in the fast path, see Fig. 5b.

**Throughput Evaluation:** For the throughput evaluation we use a constant stream of packets replayed at a specific rate. We opted for small packets to focus on the packets per second (pps) throughput rather than the bytes per second throughput. Indeed, pps throughput indicates performance bottlenecks earlier [34] than bytes per second. As in the latency experiments, we opted to use packets that are $60B$ long. Each experimental run lasts for 1000 seconds and uses a specific replay rate. Then we reset the system and start with the next replay rate. Our evaluation focuses on the last 900 seconds. For the slow path, the replay rates start from $10k$ to $40k$ packets per second, in steps of $1k$ pps. For the fast path, the replay rates start from $300k$ to $900k$ packets per second, in steps of $10k$ pps. For better readability we show the slow path plot from $10k$ to $35k$ pps.

An overview of the results for the slow and fast path throughput measurements are depicted in Figures 5c and 5d resp. In the slow path, packet loss for the vanilla kernel first sets in just after $18k$ pps, while the experiments on the grsecurity enabled kernel already exhibit packet loss at $14k$ pps. In the fast path, grsec exhibits packet loss from $350k$ pps whereas the vanilla kernel starts to drop packets at $690k$ pps. Hence, we note that the grsecurity kernel patch does have a measurable impact on the forwarding throughput in the slow and fast path of OvS. With respect to the user-land security features, we observe an overhead only in the slow path of approximately 4-15%.

**Summary:** Our measurements demonstrate that user-land mitigations do not have a large impact on OvS's forwarding performance. However, grsecurity kernel patches do cause a performance overhead for latency as well as throughput. Given that cloud systems support a variety of workloads, e.g., low latency or high throughput, kernel-based mitigations may or may not be used. However, cloud systems such as the one studied by Pfaff et al. [64] can adopt the user-land and kernel software mitigations described in this paper.

It is only a question of time until the next wormable vulnerability in a virtual switch is discovered. As software mitigations can be more easily deployed than a fully re-designed virtual switch ecosystem, we strongly recommend the adoption of software countermeasures, until a more securely designed virtual switch platform can be rolled out.

Moreover, our security analysis underlines the need for networking researchers to include software countermeasures in their design, implementation, and evaluation of novel networking components. As indicated by our analysis of related virtual switch network technologies, the networking research community must integrate security considerations into their work on new SDN and NFV technologies.

# 7 DESIGN COUNTERMEASURES

Specific attacks against virtual switches may be prevented by software countermeasures. However, the underlying problems of co-location and a worm-friendly system design remain. Hence, in this section, we present mitigation strategies that detect, isolate, and prevent the spread of attacks via the data plane and, thus, reduce the attack surface we identified. We do so not only for cloud based systems and OvS but also in the more general context of SDN.

**Virtualized/Isolated data plane:** One essential feature of the identified attack surface is the co-location of data plane and hypervisor (see Section 3). Addressing this problem in OpenStack is non-trivial due to the sheer number of interacting components and possible configurations, e.g., virtualized/non-virtualized, integrated/distributed, redundant/hierarchical controllers [69].

One way to design a system with stronger separation is to virtualize the data plane components, thereby de-coupling it from the virtualization layer. For virtual switches one example of such a proposal is to shift the position of the virtual switch from the host to a dedicated guest as proposed by Jin et al. [37]. However, the IOMMU of the host must be used to restrict access of the network cards to the network interfaces. Otherwise the physical host and the operating system running there are left vulnerable to direct memory access (DMA) attacks [86]. Such a design reduces the host OS's Trusted Computing Base (TCB) and, thereby, the attack surface of the virtual switch. We note that Arrakis [59] and IX [12] are promising proposals for HPFPs that would allow for designing such a system. Note, that while Arrakis utilizes the IOMMU, the authors of IX left this for further work.

Furthermore, to reduce the attack surface of hypervisors, Szefer et al. [87] suggest that the hypervisor should disengage itself from guest VMs, and the VM should receive direct access to the hardware (e.g., NIC). In conjunction with our suggestion of transferring the virtual switch into a virtual machine, the approach of Szefer et al. results in a more secure data plane that can no longer attack the hypervisor.

**Control plane communication firewalls:** Another method to contain and prevent attacks like the worm is tight firewalling of the control plane. In contrast to "normal" Internet traffic, control plane traffic has characteristics that enable a tighter and more secure firewall design: (i) The control plane *traffic volume* should be *significantly smaller* than regular network traffic. (ii) Nodes should only *communicate via the controller* and *not among each other*. Hence, there is a *central location* for the firewall. (iii) On the control channel there should *only* be the *control communication protocol*, e.g., the OpenFlow protocol. Even if more protocols are necessary, e.g., Simple Network Management Protocol (SNMP), the list is small, *favoring a white-listing approach*. (iv) The

*communication protocol* for SDN systems is *clearly defined*. Hence, in addition to the networking layer checks a *strict syntactic white-listing* of the control messages is feasible.

Thus, implementing a firewall and/or IDS that intercepts and cleans all control communication appears feasible. Depending on the threat model, one may even opt to chain multiple IDS/firewalls or use physical appliances for such firewalling [24].

## 8 RELATED WORK

**Cloud systems:** In the past, various attacks on cloud systems have been demonstrated. Ristenpart et al. [70] show how an attacker can co-locate her VM with a target VM to obtain secret information. Costin et al. [17] find vulnerabilities in web-based interfaces operated by cloud providers. Wu et al. [96] assess the network security of VMs in computing clouds. They point out what sniffing and spoofing attacks a VM can carry out in a virtual network. Ristov et al. [71] investigate the security of a default *OpenStack* deployment and show that it is vulnerable from the inside rather than the outside. Indeed, the *OpenStack* security guide [6] mentions that *OpenStack* is inherently vulnerable to insider threats due to bridged domains (Public and Management APIs, Data and Management, etc.).

**SDN security:** Several researchers have pointed out security threats for SDN. For example, Klöti et al. [42] report on STRIDE, a threat analysis of OpenFlow, and Kreutz et al. [45] survey several threat vectors that may enable the exploitation of SDN vulnerabilities.

So far, work on how to handle malicious switches is sparse. Sonchack et al. describe a framework for enabling practical software-defined networking security applications [85] and Shin et al. [83] present a flow management system for handling malicious switches. Work on compromised data planes is sparse as well. For example, Matsumoto et al. [50] focus on insider threats. Furthermore, national security agencies are reported to have bugged networking equipment [5] and networking vendors have left backdoors open [3, 4, 15], leading to additional threats.

Hong et al. [30] focus on how the controller's view of the network (topology) can be compromised. They identify topology based attacks in an SDN that allow an attacker to create false links to perform man-in-the-middle and blackhole attacks. Although they discovered novel SDN attacks, their threat model does not account for a compromised data plane.

**Data plane security:** Lee et al. [47] investigate how malicious routers can disrupt data plane operations, while Kamisinski et al. [39] demonstrate methods to detect malicious switches in an SDN. In addition, Porez-Botero et al. [58] characterize possible hypervisor vulnerabilities and identify Network/IO as one. In contrast to our work, they omit a deep analysis on the challenges introduced by co-located data planes. Hence, they did not find any network based vulnerabilities. Dobrescu et al. [21] develop a data plane verification tool for the *Click* software. They prove properties such as crash-freedom, bounded execution, or filtering correctness for the switch's data plane. Although software verification can ensure the correctness and security of greenfield software data plane solutions, they currently fall short of ensuring this for legacy software. In such a scenario, coverage guided fuzz testing is a more appropriate approach.

## 9 CONCLUDING REMARKS

In this paper we present our study of the attack surface of today's virtualized data planes as they are frequently used in SDN-based cloud systems. We demonstrate that virtual switches are susceptible to various attacks by *design*. Furthermore, we point out that existing threat models for virtual switches are insufficient. Accordingly, we derive a new attacker model for virtual switches and underline this by demonstrating a successful attack against OpenStack.

Our survey of related data plane technologies including NFV/SDN and other virtual switches finds that they are susceptible to the same security design flaws. We find that readily available software security measures are commonly not evaluated for new data plane components. This is unfortunate, as our evaluation of such techniques indicates that they introduce minor performance overheads in user-space.

With hardware vendors, e.g., Broadcom, selling so-called *SmartNICs* [27, 56], i.e., NICs running a full fledged virtual switch such as OvS, we believe the attack surface has been extended to the NIC as well. As we demonstrated, neglecting security during the design of virtual switches, SDN, and, NFV data plane components can have dramatic consequences on deployed real-world systems.

## ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their valuable feedback and comments. The authors would like to express their gratitude towards the German *Federal Office for Information Security*, for initial discussions on the security of the SDN data plane. This work was partially supported by the Helmholtz Research School in Security Technologies scholarship, Danish Villum Foundation project "ReNet", BMBF Grant KIS1DSD032 (Project Enzevalos), the "API Assistant" activity of EIT Digital, 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. Finally, we thank Jan Nordholz, Julian Vetter and Robert Buhren for their valuable discussions on the software countermeasures.

## REFERENCES


[1] [n. d.]. ROPGadget Tool. https://github.com/JonathanSalwan/ROPgadget/tree/master. ([n. d.]). Accessed: 02-06-2016.

[2] 2009. Cisco VN-Link: Virtualization-Aware Networking. White paper. (2009).

[3] 2013. Huawei HG8245 backdoor and remote access. http://websec.ca/advisories/view/Huawei-web-backdoor-and-remote-access. (2013). Accessed: 27-01-2017.

[4] 2014. Netis Routers Leave Wide Open Backdoor. http://blog.trendmicro.com/trendlabs-security-intelligence/netis-routers-leave-wide-open-backdoor/. (2014). Accessed: 27-01-2017.

[5] 2014. Snowden: The NSA planted backdoors in Cisco products. http://www.infoworld.com/article/2608141/internet-privacy/snowden--the-nsa-planted\-backdoors-in-cisco-products.html. (2014). Accessed: 27-01-2017.

[6] 2016. OpenStack Security Guide. http://docs.openstack.org/security-guide. (2016). Accessed: 27-01-2017.

[7] 2016. What is Openstack? https://www.openstack.org/software. (2016).

[8] Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. 2005. Control-flow Integrity. In *Proc. ACM Conference on Computer and Communications Security (CCS)*. 340–353.

[9] Nawaf Alhebaishi, Lingyu Wang, Sushil Jajodia, and Anoop Singhal. 2016. Threat Modeling for Cloud Data Center Infrastructures. In *Intl. Symposium on Foundations and Practice of Security*. Springer, 302–319.

[10] L. Andersson, P. Doolan, N. Feldman, A. Fredette, and B. Thomas. 2001. LDP Specification. RFC 3036 (Proposed Standard). (January 2001). http://www.ietf.org/rfc/rfc3036.txt Obsoleted by RFC 5036.

[11] Hitesh Ballani, Paolo Costa, Christos Gkantsidis, Matthew P Grosvenor, Thomas Karagiannis, Lazaros Koromilas, and Greg O'Shea. 2015. Enabling end-host network functions. In *ACM Computer Communication Review (CCR)*, Vol. 45. 493–507.

[12] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. 2014. IX: A protected dataplane operating system for high throughput and low latency. In *Proc. Usenix Symposium on Operating Systems Design and Implementation (OSDI)*. 49–65.

[13] BESS Comitters. 2017. BESS (Berkeley Extensible Software Switch). https://github.com/NetSys/bess. (2017). Accessed: 09-05-2017.

[14] Martín Casado, Teemu Koponen, Rajiv Ramanathan, and Scott Shenker. 2010. Virtualizing the Network Forwarding Plane. In *Proc. ACM CoNEXT Workshop on Programmable Routers for Extensible Services of Tomorrow*. Article 8, 6 pages.

[15] Stephen Checkoway et al. 2016. A Systematic Analysis of the Juniper Dual EC Incident. Cryptology ePrint Archive, Report 2016/376. (2016).

[16] Sean Choi, Xiang Long, Muhammad Shahbaz, Skip Booth, Andy Keep, John Marshall, and Changhoon Kim. 2017. PVPP: A Programmable Vector Packet Processor. In *Proc. ACM Symposium on Software Defined Networking Research (SOSR)*. ACM, 197–198.

[17] Andrei Costin. 2015. All your cluster-grids are belong to us: Monitoring the (in)security of infrastructure monitoring systems. In *Proc. IEEE Communications and Network Security (CNS)*. 550–558. https://doi.org/10.1109/CNS.2015.7346868

[18] Crispin Cowan et al. 1998. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-overflow Attacks. In *Proc. Usenix Security Symp.* 5–5.

[19] Ana Danping et al. 2016. Threat Analysis for the SDN Architecture. Technical Report. (2016).

[20] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. 2015. SPHINX: Detecting Security Attacks in Software-Defined Networks.. In *Proc. Internet Society Symposium on Network and Distributed System Security (NDSS)*.

[21] Mihai Dobrescu and Katerina Argyraki. 2014. Software Dataplane Verification. In *Proc. Usenix Symposium on Networked Systems Design and Implementation (NSDI)*. 101–114.

[22] Dan Gonzales et al. 2017. Cloud-Trust - a Security Assessment Model for Infrastructure as a Service (IaaS) Clouds. *Proc. IEEE Conference on Cloud Computing* PP, 99 (2017), 1–1.

[23] Nick Feamster, Jennifer Rexford, and Ellen Zegura. 2013. The Road to SDN. *Queue* 11, 12 (December 2013).

[24] Anja Feldmann, Philipp Heyder, Michael Kreutzer, Stefan Schmid, Jean-Pierre Seifert, Haya Shulman, Kashyap Thimmaraju, Michael Waidner, and Jens Sieberg. 2016. NetCo: Reliable Routing With Unreliable Routers. In *IEEE Workshop on Dependability Issues on SDN and NFV*.

[25] Tobias Fiebig, Franziska Lichtblau, Florian Streibelt, Thorben Krueger, Pieter Lexis, Randy Bush, and Anja Feldmann. 2016. SoK: An Analysis of Protocol Design: Avoiding Traps for Implementation and Deployment. *arXiv preprint arXiv:1610.05531* (2016).

[26] Daniel Firestone. 2017. VFP: A Virtual Switch Platform for Host SDN in the Public Cloud.. In *Proc. Usenix Symposium on Networked Systems Design and Implementation (NSDI)*. 315–328.

[27] Andy Gospodarek. 2017. The birth of SmartNICs – offloading dataplane traffic to...software. https://youtu.be/AGSy51VlKaM. (2017). Open vSwitch Fall Conference 2017. Accessed: 29-01-2018.

[28] Bernd Grobauer, Tobias Walloschek, and Elmar Stocker. 2011. Understanding Cloud Computing Vulnerabilities. *Proc. IEEE Security & Privacy (S&P)* 9, 2 (March 2011), 50–57. https://doi.org/10.1109/MSP.2010.115

[29] Michio Honda, Felipe Huici, Giuseppe Lettieri, and Luigi Rizzo. 2015. mSwitch: a highly-scalable, modular software switch. In *Proc. ACM Symposium on Software Defined Networking Research (SOSR)*. 1.

[30] Sungmin Hong, Lei Xu, Haopei Wang, and Guofei Gu. 2015. Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures.. In *Proc. Internet Society Symposium on Network and Distributed System Security (NDSS)*.

[31] Jinho Hwang, KK Ramakrishnan, and Timothy Wood. 2014. NetVM: high performance and flexible networking using virtualization on commodity platforms. In *Proc. Usenix Symposium on Networked Systems Design and Implementation (NSDI)*. 445–458.

[32] Intel. 2015. Enabling NFV to Deliver on its Promise. https://www-ssl.intel.com/content/www/us/en/communications/nfv-packet-processing-brief.html. (2015).

[33] Ethan J Jackson et al. 2016. Softflow: A middlebox architecture for open vswitch. In *Usenix Annual Technical Conference (ATC)*. 15–28.

[34] Van Jacobson. 1988. Congestion avoidance and control. In *ACM Computer Communication Review (CCR)*, Vol. 18. 314–329.

[35] Sushant Jain et al. 2013. B4: Experience with a Globally-deployed Software Defined Wan. In *Proc. ACM SIGCOMM*. 3–14.

[36] Samuel Jero et al. 2017. BEADS: Automated Attack Discovery in OpenFlow-Based SDN Systems. In *Proc. RAID Recent Advances in Intrusion Detection*.

[37] Xin Jin, Eric Keller, and Jennifer Rexford. 2012. Virtual Switching Without a Hypervisor for a More Secure Cloud. San Jose, CA.

[38] Daya Kamath et al. 2010. Edge virtual Bridge Proposal, Version 0. Rev. 0.1. *Apr* 23 (2010), 1–72.

[39] Andrzej Kamisiński and Carol Fung. 2015. FlowMon: Detecting Malicious Switches in Software-Defined Networks. In *Proc. ACM Workshop on Automated Decision making for Active Cyber Defense*. 39–45.

[40] Kallol Krishna Karmakar, Vijay Varadharajan, and Uday Tupakula. 2017. Mitigating attacks in Software Defined Network (SDN). 112–117.

[41] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header Space Analysis: Static Checking for Networks. In *Proc. Usenix Symposium on Networked Systems Design and Implementation (NSDI)*.

113–126.

[42] R. Klöti, V. Kotronis, and P. Smith. 2013. OpenFlow: A security analysis. In *Proc. IEEE International Conference on Network Protocols (ICNP)*. 1–6. https://doi.org/10.1109/ICNP.2013.6733671

[43] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. 2000. The Click modular router. *ACM Trans. Computer Systems* 18, 3 (2000), 263–297.

[44] Teemu Koponen et al. 2014. Network Virtualization in Multi-tenant Datacenters. In *Proc. Usenix Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 203–216.

[45] Diego Kreutz, Fernando M.V. Ramos, and Paulo Verissimo. 2013. Towards Secure and Dependable Software-defined Networks. In *Proc. ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. 55–60.

[46] Volodymyr Kuznetsov et al. 2014. Code-Pointer Integrity. In *Proc. Usenix Symposium on Operating Systems Design and Implementation (OSDI)*. 147–163.

[47] Sihyung Lee, Tina Wong, and Hyong S Kim. 2006. Secure split assignment trajectory sampling: A malicious router detection system. In *Proc. IEEE/IFIP Transactions on Dependable and Secure Computing (DSN)*. 333–342.

[48] Joao Martins et al. 2014. ClickOS and the Art of Network Function Virtualization. In *Proc. Usenix Symposium on Networked Systems Design and Implementation (NSDI)*. 459–473.

[49] Craig Matsumoto. 2014. 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/. (2014). Accessed: 27-01-2017.

[50] Stephanos Matsumoto, Samuel Hitz, and Adrian Perrig. 2014. Fleet: Defending SDNs from malicious administrators. In *Proc. ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. 103–108.

[51] Nick McKeown et al. 2008. OpenFlow: enabling innovation in campus networks. *ACM Computer Communication Review (CCR)* 38, 2 (2008), 69–74.

[52] Microsoft. 2013. Hyper-V Virtual Switch Overview. https://technet.microsoft.com/en-us/library/hh831823(v=ws.11).aspx. (2013). Accessed: 27-01-2017.

[53] László Molnár, Gergely Pongrácz, Gábor Enyedi, Zoltán Lajos Kis, Levente Csikor, Ferenc Juhász, Attila Kőrösi, and Gábor Rétvári. 2016. Dataplane Specialization for High-performance OpenFlow Software Switching. In *Proc. ACM SIGCOMM*. 539–552.

[54] Yoshihiro Nakajima, Tomoya Hibi, Hirokazu Takahashi, Hitoshi Masutani, Katsuhiro Shimano, and Masaki Fukui. 2014. Scalable high-performance elastic software OpenFlow switch in userspace for wide-area network. *USENIX Open Networking Summit* (2014), 1–2.

[55] Nicolae Paladi and Christian Gehrmann. 2015. Towards Secure Multi-tenant Virtualized Networks, Vol. 1. 1180–1185.

[56] Manoj Panicker. 2017. Enabling Hardware Offload of OVS Control & Data plane using LiquidIO. https://youtu.be/qjXBRCFhbqU. (2017). Open vSwitch Fall Conference 2017. Accessed: 29-01-2018.

[57] Mathias Payer. 2012. Too much PIE is bad for performance. http://e-collection.library.ethz.ch/eserv/eth:5699/eth-5699-01.pdf. (2012). Accessed: 27-01-2017.

[58] Diego Perez-Botero, Jakub Szefer, and Ruby B. Lee. 2013. Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers. In *Proc. ACM Workshop on Security in Cloud Computing*. 3–10.

[59] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. 2014. Arrakis: The Operating System is the Control Plane. In *Proc. Usenix Symposium on Operating Systems Design and Implementation (OSDI)*. 1–16.

[60] Benjamin D Peterson. 2012. Security Implications of the Cisco Nexus 1000V. http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1069&context=techmasters. (2012). Accessed: 27-01-2017.

[61] Justin Pettit, Ben Pfaff, Chris Wright, and Madhu Venugopal. 2015. OVN, Bringing Native Virtual Networking to OVS. https://networkheresy.com/2015/01/13/ovn-bringing-native-virtual-networking-to-ovs/. (2015). Accessed: 27-01-2017.

[62] Ben Pfaff. 2013. Open vSwitch: Past, Present, and Future. http://openvswitch.org/slides/ppf.pdf. (2013). Accessed: 27-01-2017.

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

[64] Ben Pfaff et al. 2015. The design and implementation of Open vSwitch. In *Proc. Usenix Symposium on Networked Systems Design and Implementation (NSDI)*. 117–130.

[65] Gregory Pickett. 2014. Abusing software defined networks. *Black Hat EU* (2014).

[66] Gergely Pongrácz, László Molnár, and Zoltán Lajos Kis. 2013. Removing roadblocks from SDN: OpenFlow software switch performance on Intel DPDK. In *European Workshop on Software Defined Networking*. IEEE, 62–67.

[67] prweb. 2011. 6WIND Extends Portable Packet Processing Software to Support Intel Data Plane Development Kit. http://www.prweb.com/releases/2011/9/prweb8785683.htm. (2011). Accessed: 27-01-2017.

[68] Kaushik Kumar Ram, Alan L Cox, Mehul Chadha, Scott Rixner, and TW Barr. 2013. Hyper-Switch: A Scalable Software Virtual Switching Architecture.. In *Usenix Annual Technical Conference (ATC)*. 13–24.

[69] Sridhar Rao. 2015. SDN's Scale Out Effect on OpenStack Neutron. http://thenewstack.io/sdn-controllers-and-openstack-part1/. (2015). Accessed: 27-01-2017.

[70] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *Proc. ACM Conference on Computer and Communications Security (CCS)*. 199–212.

[71] Sasko Ristov, Marjan Gusev, and Aleksandar Donevski. 2013. Openstack cloud security vulnerabilities from inside and outside. Technical Report. (2013), 101–107 pages.

[72] Luigi Rizzo. 2012. Netmap: a novel framework for fast packet I/O. In *Usenix Annual Technical Conference (ATC)*. 101–112.

[73] Luigi Rizzo and Giuseppe Lettieri. 2012. VALE, a Switched Ethernet for Virtual Machines. In *Proc. ACM CoNEXT*. 61–72.

[74] Robin G. 2016. Open vSwitch with DPDK Overview. https://software.intel.com/en-us/articles/open-vswitch-with-dpdk-overview. (2016). Accessed: 27-01-2017.

[75] Ryan Roemer, Erik Buchanan, Hovav Shacham, and Stefan Savage. 2012. Return-Oriented Programming: Systems, Languages, and Applications. *ACM Trans. on Information and System Security (TISSEC)* 15, 1, Article 2 (March 2012), 34 pages.

[76] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, and A. Conta. 2001. MPLS Label Stack Encoding. RFC 3032 (Proposed Standard). (January 2001). http://www.ietf.org/rfc/rfc3032.txt Updated by RFCs 3443, 4182, 5332, 3270, 5129, 5462, 5586, 7274.

[77] E. Rosen, A. Viswanathan, and R. Callon. 2001. Multiprotocol Label Switching Architecture. RFC 3031 (Proposed Standard). (January 2001). http://www.ietf.org/rfc/rfc3031.txt Updated by RFCs 6178, 6790.

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

[79] Len Sassaman, M. L. Patterson, S. Bratus, and M. E. Locasto. 2013. Security Applications of Formal Language Theory. 7, 3 (Sept 2013), 489–500.

[80] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, and Jennifer Rexford. 2016. Pisces: A programmable, protocol-independent software switch. In *Proc. ACM SIGCOMM*. 525–538.

[81] Bhargava Shastry. 2017. Fuzzing Open vSwitch. https://bshastry.github.io/2017/07/24/Fuzzing-OpenvSwitch.html. (2017). Accessed: 29-01-2018.

[82] Bhargava Shastry, Markus Leutner, Tobias Fiebig, Kashyap Thimmaraju, Fabian Yamaguchi, Konrad Rieck, Jean-Pierre Seifert Stefan Schmid, and Anja Feldmann. 2017. Static Program Analysis as a Fuzzing Aid. In *Proc. RAID Recent Advances in Intrusion Detection*.

[83] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. 2013. AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-defined Networks. In *Proc. ACM Conference on Computer and Communications Security (CCS)*. 413–424.

[84] Pontus Sköldström, Balázs Sonkoly, András Gulyás, Felicián Németh, Mario Kind, Fritz-Joachim Westphal, Wolfgang John, Jokin Garay, Eduardo Jacob, Dávid Jocha, et al. 2014. Towards Unified Programmability of Cloud and Carrier Infrastructure. In *European Workshop on Software Defined Networking*. 55–60.

[85] John Sonchack, Adam J Aviv, Eric Keller, and Jonathan M Smith. 2016. Enabling Practical Software-defined Networking Security Applications with OFX. In *Proc. Internet Society Symposium on Network and Distributed System Security (NDSS)*.

[86] Patrick Stewin. 2013. A primitive for revealing stealthy peripheral-based attacks on the computing platform's main memory. In *Proc. RAID Recent Advances in Intrusion Detection*. Springer, 1–20.

[87] Jakub Szefer et al. 2011. Eliminating the Hypervisor Attack Surface for a More Secure Cloud. In *Proc. ACM Conference on Computer and Communications Security (CCS)*. 401–412.

[88] T. Koponen et al. 2014. Network Virtualization in Multi-tenant Datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation*.

[89] Kashyap Thimmaraju et al. 2017. The vAMP Attack: Taking Control of Cloud Systems via the Unified Packet Parser. In *Proc. ACM Workshop on Cloud Computing Security (CCSW)*.

[90] Kashyap Thimmaraju, Liron Schiff, and Stefan Schmid. 2017. Outsmarting Network Security with SDN Teleportation. In *Proc. IEEE European Security & Privacy (S&P)*.

[91] Heidi Joy Tretheway et al. Apr 2016. A snapshot of Openstack users' attitudes and deployments. *Openstack User Survey* (Apr 2016).

[92] Amin Vahdat. 2014. Enter the Andromeda zone - Google Cloud Platform's latest networking stack. https://cloudplatform.googleblog.com/2014/04/enter-andromeda-zone-google-cloud-platforms-latest-networking-stack.html. (2014). Accessed: 19-10-2017.

[93] Rick Vanover. 2008. Virtual switching to become enhanced with Cisco and VMware announcement. http://www.techrepublic.com/blog/data-center/virtual-switching-to-become-enhanced-with-cisco-and-vmware-announcement. (2008). Accessed: 27-01-2017.

[94] VMware. 2009. VMware ESX 4.0 Update 1 Release Notes. https://www.vmware.com/support/vsphere4/doc/vsp$_{e}$sx40$_{u}$1$_{r}$el$_{n}$otes.html. (2009). Accessed: 27-01-2017.

[95] VPP Comitters. 2017. What is VPP? https://wiki.fd.io/view/VPP/What$_{i}$s$_{V}$PP%3F. (2017). Accessed: 09-05-2017.

[96] Hanqian Wu et al. 2010. Network security for virtual machine in cloud computing. In *Proc. IEEE Conference on Computer Sciences and Convergence Information Technology*. 18–21.

[97] Dongting Yu, Andrew W Moore, Chris Hall, and Ross Anderson. 2014. *Security: a Killer App for SDN?* Technical Report. Indiana Uni. at Bloomington.