The vAMP Attack: Compromising Cloud Systems via the Unified Packet Parser

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

CCSW'17, Dallas, Texas, USA
3 Nov. 2017

Multi-tenant IaaS cloud providers

Google

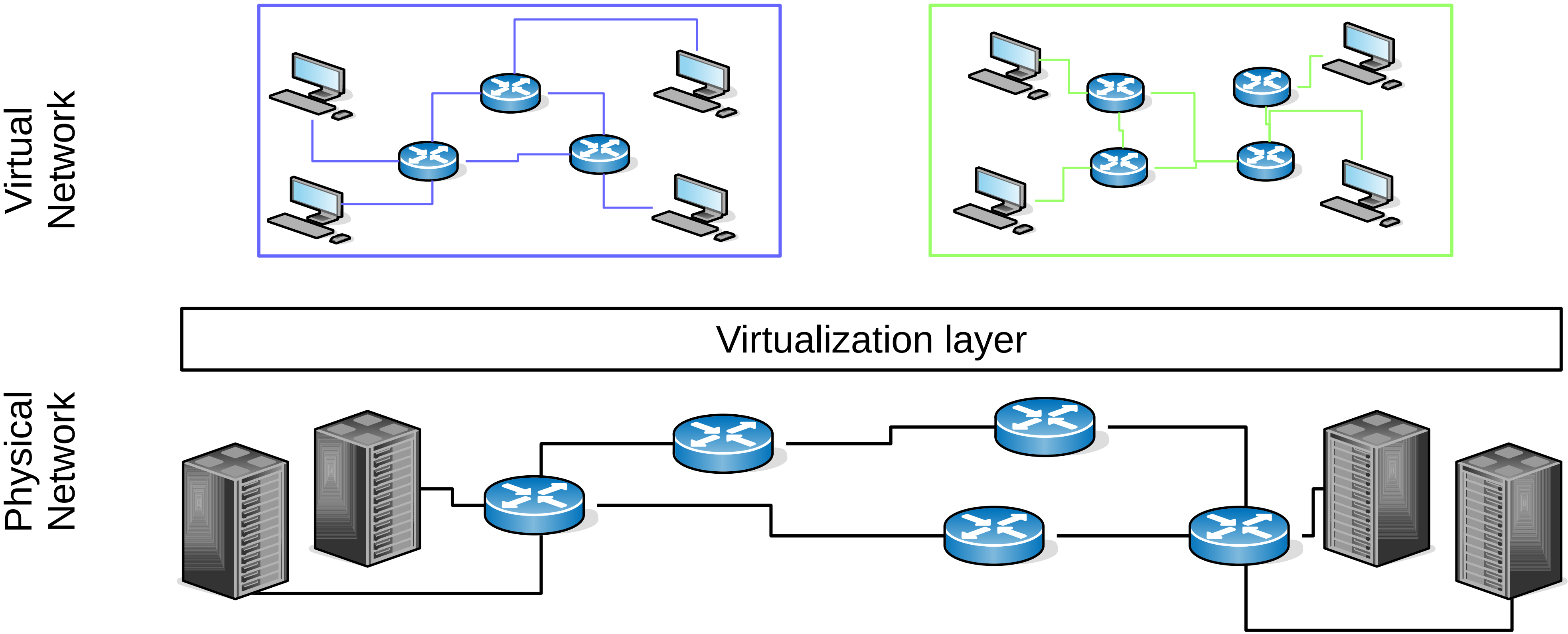
Microsoft Azure

amazon web services

Key enabler for multi-tenancy is virtualization

Compute	**Network**	Storage
Full	?	Block
Para		File

What is network virtualization?



Key enabler for multi-tenancy is virtualization

Compute	**Network**	Storage
Full	Virtual switches	Block
Para		File

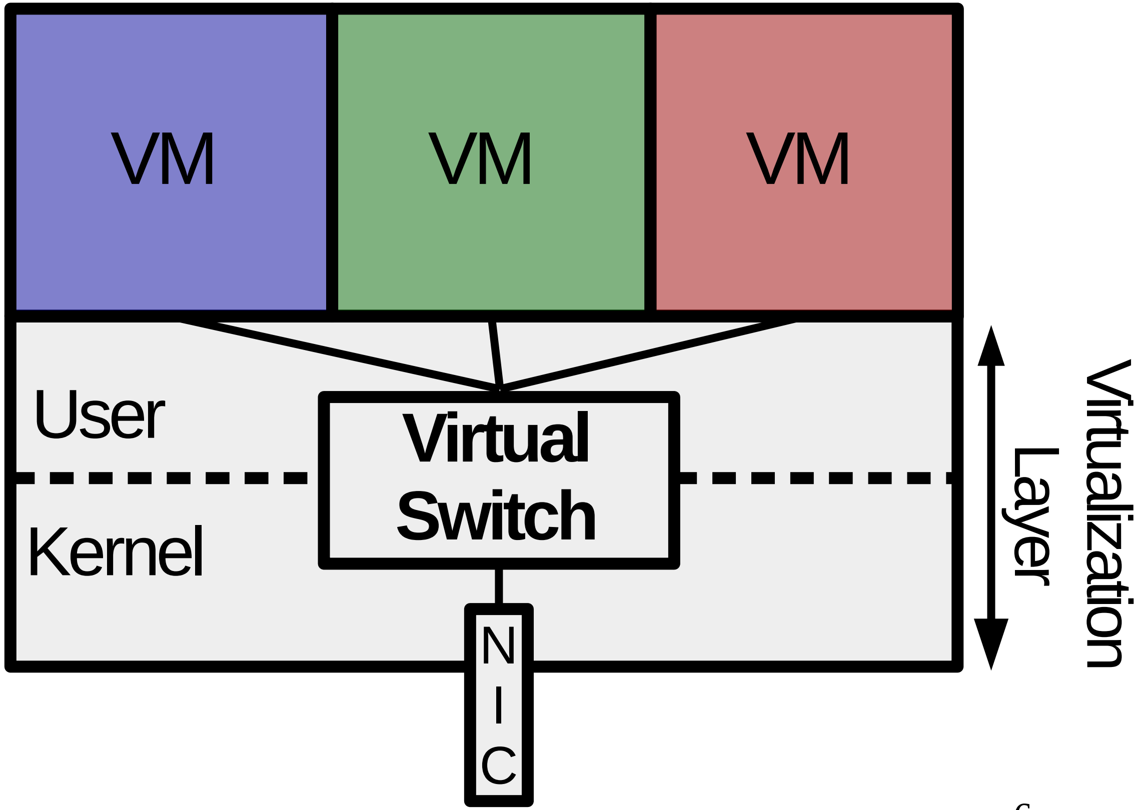
vmware®

OvS Open vSwitch

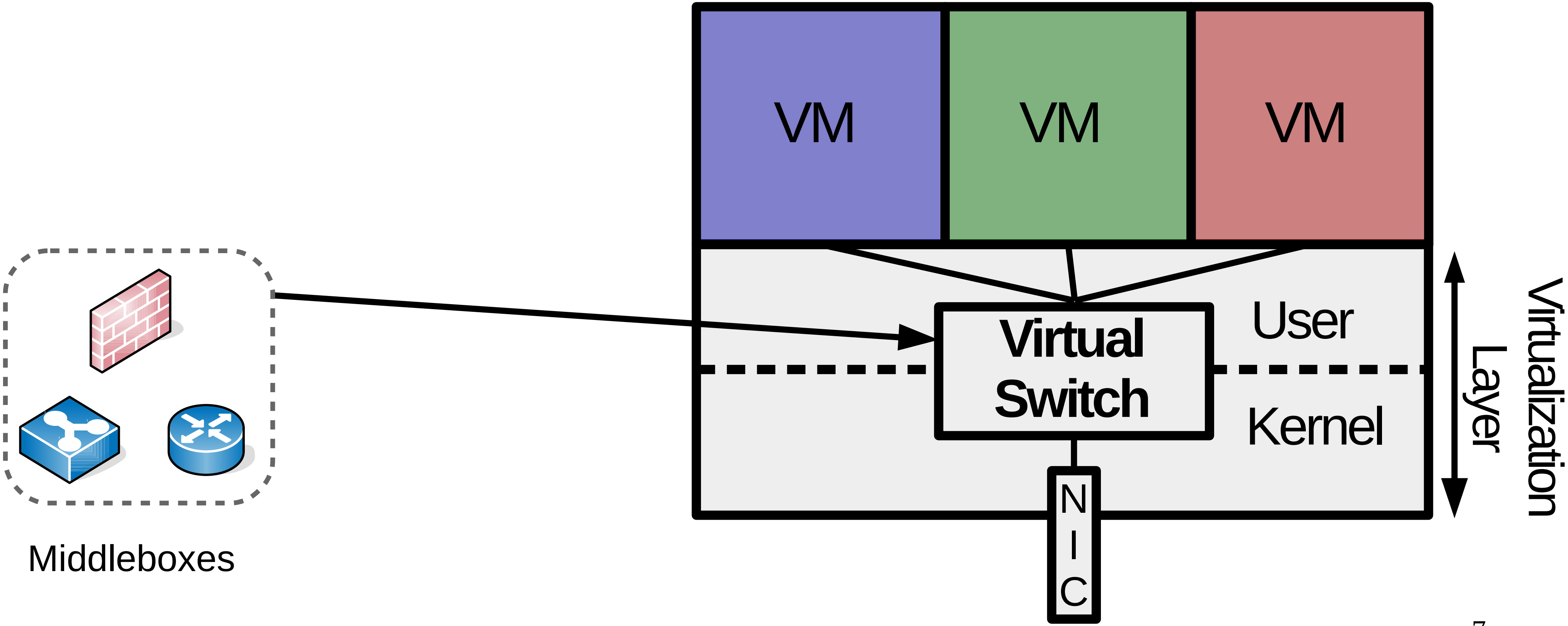
cisco

Virtual switches: The network hypervisor

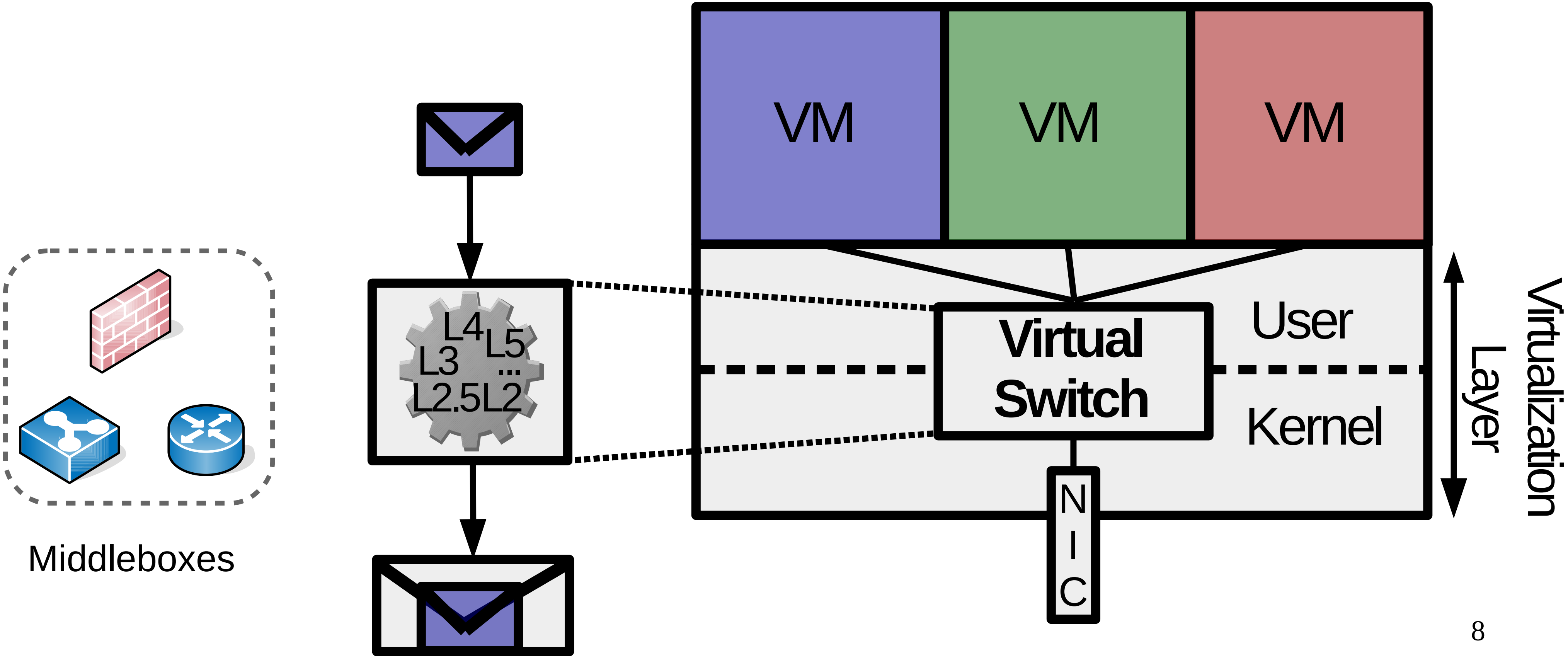
- Meant to provide *network isolation*
- Centralized control
- Programmable



Introducing (complex) network functionality into the virtual switch



Results in a lot of packet parsing in the virtual switch

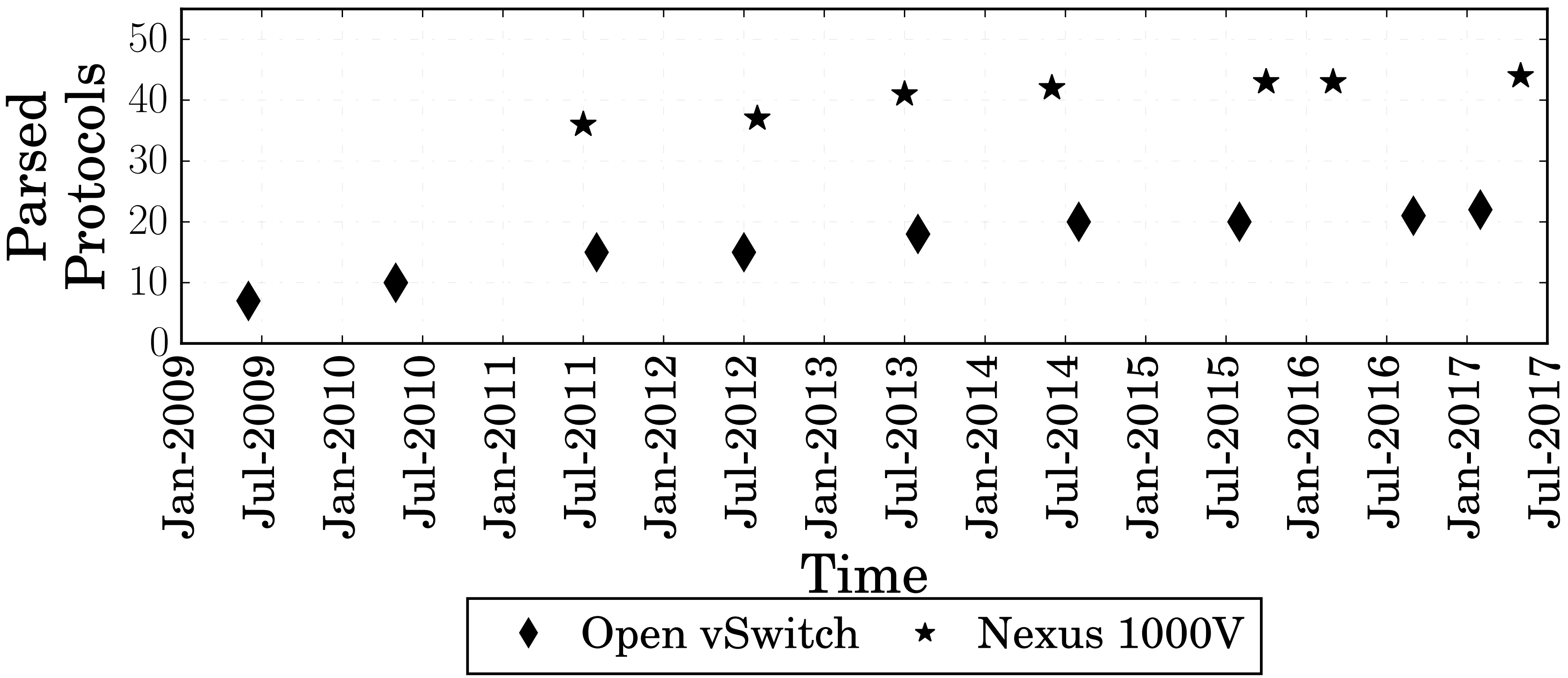


The unified packet parser: A new attack surface for virtual switches

- *Centralized parsing* in the virtual switch, i.e., parse all the headers of a packet in a single pass
- *Error prone* as parsing logic is implemented manually
- Dependent security mechanisms and policies can be bypassed if broken

Open vSwitch Protocols
Ethernet
LLC
VLAN
MPLS
IPv4
ICMPv4
TCP
UDP
ARP
SCTP
IPv6
ICMPv6
IPv6 ND
GRE
LISP
VXLAN
PBB
IPv6 EXT HDR
TUNNEL-ID
IPv6 ND
IPv6 EXT HDR
IPv6HOPOPTS
IPv6ROUTING
IPv6Fragment
IPv6DESTOPT
IPv6ESP
IPv6 AH
RARP
IGMP

Supported protocols in OvS and Cisco Nexus 1000V over time



Let's look at threat/attacker models
for virtual switches

Previous models (non-exhaustive)

- General, for the data plane
  - Chasaki et al. [1]
  - Keller et al. [2]
  - Qubes OS [3]
  - Dhawan et al. [4]
- Strong adversary, for hardware switches
  - Yu et al. [11]
  - Thimmaraju et al. [12]
- Conservative, for network virtualization
  - Paladi et al. [5]
  - Grobauer et al. [6]
- Underestimated, for virtual switches
  - Jin et al. [7]
  - Alhebaishi et al. [8]
  - Gonzales et al. [9]
  - Karmaker et al. [10]

Attacker Model

- **Attacker**
  - Limited resources/Lone wolf
  - No vantage point access
  - Avg. programming languages skills
  - Controls a computer that is publicly reachable
- **Defender**
  - Uses virtual switches for network virtualization
  - Follows cloud security best practises [13]
  - Uses the same software stack across all servers

Attack is successful if the attacker obtains full control of the cloud, i.e., perform arbitrary computation, create/store arbitrary data, and send/receive arbitrary data to all nodes

Taking control of the cloud

Attack setup

Virtual switch

Open vSwitch

Cloud management system

OpenStack

Program analyzer

American Fuzzy Lop (AFL)

Fig credits: Breadtk [15]

Attack methodology: Fuzzing

- Targeted the unified packet parser of Open vSwitch (~3% of total execution paths in ovs-vswitchd)
- Leveraged the test-flows test case
- Tested ovs-2.3.2, ovs-2.4.0 and ovs-2.5.0

- Found several vulnerabilities reported in 2 CVEs
  - *CVE-2016-2074*
    - *Remote code execution*
    - Denial of service
  - CVE-2016-10377
    - ACL bypass

CVE-2016-2074

- Problems in parsing the MPLS label stack
  - *Extremely long label stack led to a stack buffer overflow in ovs-2.3.**
  - Early terminating label stack led to a stack buffer overflow in ovs-2.3.* and ovs-2.4.0
- RFC 3032 says: Pop top label and then decide what do to
- Exploits unified packet parser: extracts all labels

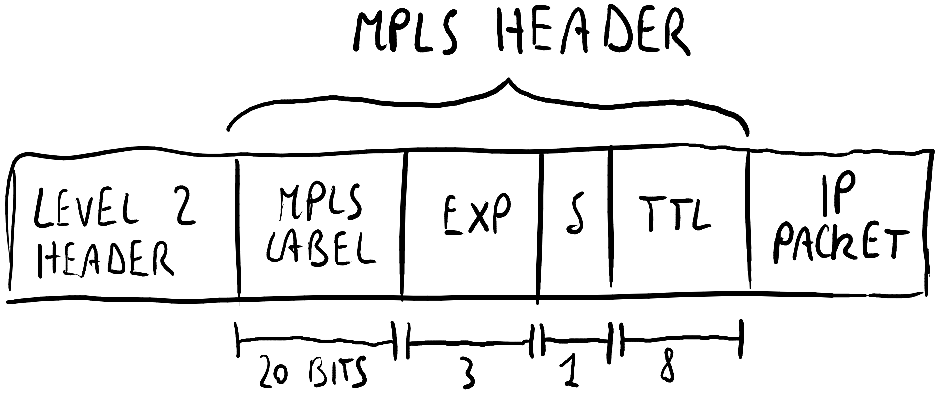
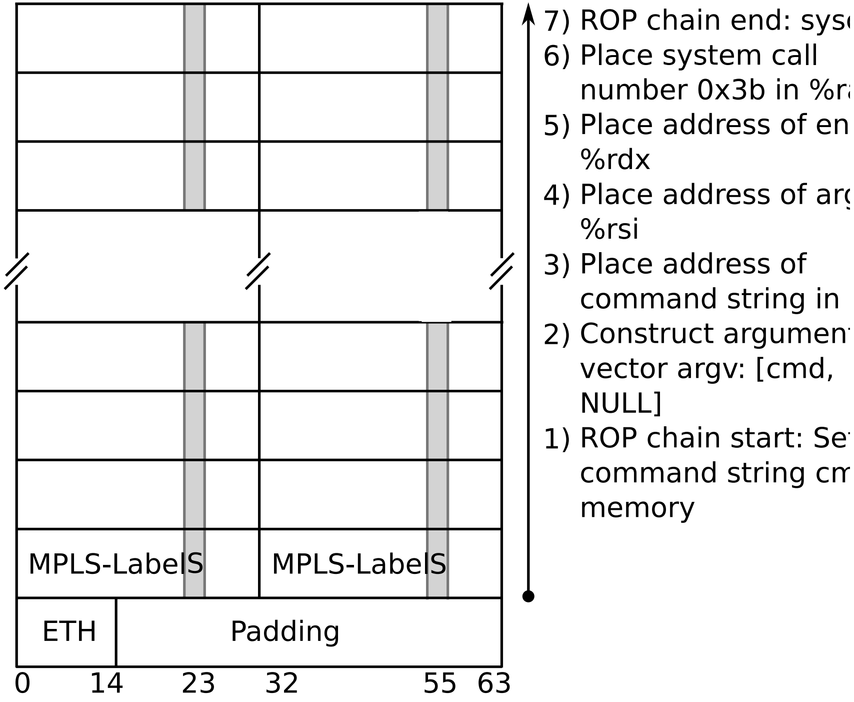


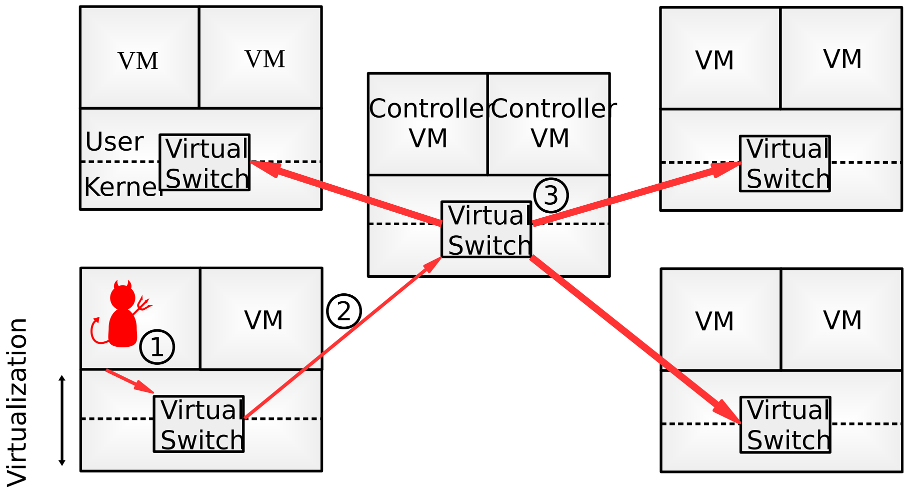
Figure credit: Lorenzo David, Luca Ghio. MPLS header [14]

Stack buffer overflow → ROP exploit

- ASLR did not help
  - No PIE by default, else code segment would have been randomized
  - All gadgets were from the ovs-vswitchd code segment as it's a fairly large binary
- Default gcc compile does not place a canary for the vulnerable function
- No sanity checks possible from the kernel/device driver



ROP exploit → Worm



- OvS had to be patched to propagate
- The exploit from the compute server to the controller server had to be adjusted due to VLAN/VXLAN encapsulation
- Required an external (to the cloud) host for command-and-control

Attack evaluation

- Used Mirantis 8.0 for setting-up OpenStack “Liberty” in VirtualBox which ships the vulnerable ovs-2.3
- 1 Compute node (VirtualBox VM) hosting 1 VM (nested virtualization!) for the attacker
- 1 Controller node (VirtualBox VM) hosting 1 VM to control the setup, and also serves as the Network node (for routing)
- Hosted the exploit for compute → controller on a publicly reachable webserver (only for testing)

Attack result

- VM→Compute → Controller : < 20s
  - 3s download, 12s sleep to restart ovs-vswitchd on compute
- Controller → other Computes : < 80s
  - 3s download, 60s sleep to restart ovs-vswitchd on controller
- Total time to own the cloud: < 2min

Conclusion

- Virtual switches implement unified packet parsers that increase the attack surface of the cloud
- We introduced the virtual switch Attacker Model for Packet-parsing (vAMP) which accounts for virtual switches in cloud systems
- We demonstrated that an entire cloud setup can be compromised in a matter of minutes by exploiting the virtual switch

Questions?

References

1. 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).
2. 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.
3. Joanna Rutkowska and Rafal Wojtczuk. “Qubes OS architecture”. In: Invisible Things Lab Tech Rep 54 (2010).
4. 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.
5. Nicolae Paladi and Christian Gehrmann. “Towards Secure Multi-tenant Virtualized Networks”. In: Proc. IEEE Trustcom/BigDataSE/ISPA. Vol. 1. Aug. 2015.
6. Bernd Grobauer, Tobias Walloschek, and Elmar Stöcker. “Understanding Cloud Computing Vulnerabilities”. In: IEEE Security & Privacy Magazine 9.2 (Mar. 2011).
7. 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
8. Nawaf Alhebaishi et al. “Threat Modeling for Cloud Data Center Infrastructures”. In: Intl. Symposium on Foundations and Practice of Security. Springer. 2016.
9. 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).
10. Kallol Krishna Karmakar, Vijay Varadharajan, and Uday Tupakula. “Mitigating attacks in Software Defined Network (SDN)”. In: Proc. IEEE Software Defined Systems (SDS). May 2017.
11. Dongting Yu et al. Security: a Killer App for SDN? Tech. rep. Indiana Uni. At Bloomington, 2014.
12. Kashyap Thimmaraju, Liron Schiff, and Stefan Schmid. “Outsmarting Network Security with SDN Teleportation”. In: Proc. IEEE European Security & Privacy (S&P). 2017.
13. OpenStack Security Guide. http://docs.openstack.org/security-guide. Accessed 27-01-2017.
14. https://commons.wikimedia.org/wiki/File:MPLS_header.svgAccessed on 26.10.2017
15. https://en.wikipedia.org/wiki/File:AFL_Fuzz_Logo.gif Accessed on 26.10.2017
16. Bhargava Shastry et al. “Static Exploration of Taint-Style Vulnerabilities Found by Fuzzing”. In Proc. USENIX Workshop on Offensive Technologies (WOOT). 2017.

Backup slides

Buggy mpls parsing function

```
/* Pulls the MPLS headers at '*datap' and returns the count of them. */
static inline int parse_mpls(void **datap, size_t *sizep)
{
    const struct mpls_hdr *mh;
    int count = 0;

    while ((mh = data_try_pull(datap, sizep, sizeof *mh))) {
        count++;
        if (mh->mpls_lse.lo & htons(1 << MPLS_BOS_SHIFT)) {
            break;
        }
    }
    return MAX(count, FLOW_MAX_MPLS_LABELS);
}
```

The function that got smashed

```
void flow_extract(struct ofpbuf *packet, const struct pkt_metadata *md,
                struct flow *flow)
{
    struct {
        struct miniflow mf;
        uint32_t buf[FLOW_U32S];
    } m;

    COVERAGE_INC(flow_extract);

    miniflow_initialize(&m.mf, m.buf);
    miniflow_extract(packet, md, &m.mf);
    miniflow_expand(&m.mf, flow);
}
```

Call hierarchy for the RCE bug

```
flow_extract(struct ofpbuf *packet, const struct pkt_metadata *md, struct flow *flow)
    …
    miniflow_extract(packet, md, &m.mf)
        ...
        count = parse_mpls(&data, &size);
        miniflow_push_words(mf, mpls_lse, mpls, count);
            miniflow_push_words_(MF, offsetof(struct flow, FIELD), VALUEP, N_WORDS)
                MINIFLOW_ASSERT(MF.data + (N_WORDS) <= MF.end && (OFS) % 4 == 0 && !(MF.map & (UINT64_MAX << ofs32)));
                memcpy(MF.data, (VALUEP), (N_WORDS) * sizeof *MF.data);
```

Ovs-2.4.0 bug: A crafted MPLS packet yields a zero 'count'

```
1. miniflow_extract():
2.         count = parse_mpls(&data, &size);
3.         miniflow_push_words_32(mf, mpls_lse, mpls, count);
```

Ovs-2.4.0 bug: miniflow_push_words_32() updated mf.map as follows:

1. mf.map |= ((UINT64_MAX >> (64 - DIV_ROUND_UP(N_WORDS, 2))) << ofs64);
2. mf.map |= (UINT64_MAX >> 64) << ofs64;

Unforunately, C renders shifting a 64-bit constant by 64 bits undefined.

On common x86 platforms, 'n << 64' is equal to 'n', so this behaves as:

3. mf.map |= UINT64_MAX << ofs64;

Ovs-2.4.0 bug: miniflow_push_words_32() updated mf.map as follows:

In this particular case, ofs64 is 15, so this sets the most-significant 48 bits of mf.map (a 63-bit bit-field) to 1. Only the least-significant 28 bits of mf.map should ever be set to 1, so this sets 35 bits to 1 that should never be. Because of the structure of the data structure that mf.map is embedded within, this makes it possible later to overwrite 8*35 == 280 bytes of data in the stack. However, there is no obvious way to control the data used in the overwrite--it is memcpy'd from one place to another but the source data does not come from the network. In the bug reporter's testing, this overwrite caused a userspace crash if debug logging was enabled, but not otherwise. This commit fixes the problem by avoiding the out-of-range shift.

ACL bypass bug: Integer underflow

- code in miniflow_extract() verified these invariants:

- size >= 20 (minimum IP header length)
- ip_len >= 20 (ditto)
- ip_len <= size (to avoid reading past end of packet)
- tot_len <= size (ditto)
- size - tot_len <= 255 (because this is stored in a 1-byte variable internally and wouldn't normally be big)

- It failed to verify the following, which is not implied by the conjunction of the above:

- ip_len <= tot_len (e.g. that the IP header fits in the packet)

More on fuzzing Open vSwitch

- Shastry et al.[16] conducted extensive fuzzing in OvS and reported several other CVEs in their WOOT'17 paper.