Policy Injection: A Cloud Dataplane DoS Attack

Levente Csikor, Christian Rothenberg, Dimitrios P. Pezaros,
Stefan Schmid, László Toka, Gábor Rétvári
University of Campinas, University of Glasgow,
University of Vienna, Budapest University of Technology and Economics

SIGCOMM 2018
BUDAPEST

Public cloud and security

- IaaS may be a double edged sword
- Promise is zero infrastructure cost, flexible resource provisioning, high availability, and usage-based pricing
- But tenants share compute, storage and network resources
- Isolation between tenants is a major worry
- Malevolent tenants may interfere with other tenants' code/data/services

Virtual switch: plausible attack target

- Creates the illusion of a per-tenant dedicated switch
- Implements the logical datapath, security policies, load-balancing, monitoring, etc., for each tenant
- Implemented on a shared hypervisor switch
- Threat model: a malevolent tenant can exhaust a shared resource in the hypervisor denying network service to the rest of the tenants

Policy injection: algorithmic complexity attack on the cloud data plane

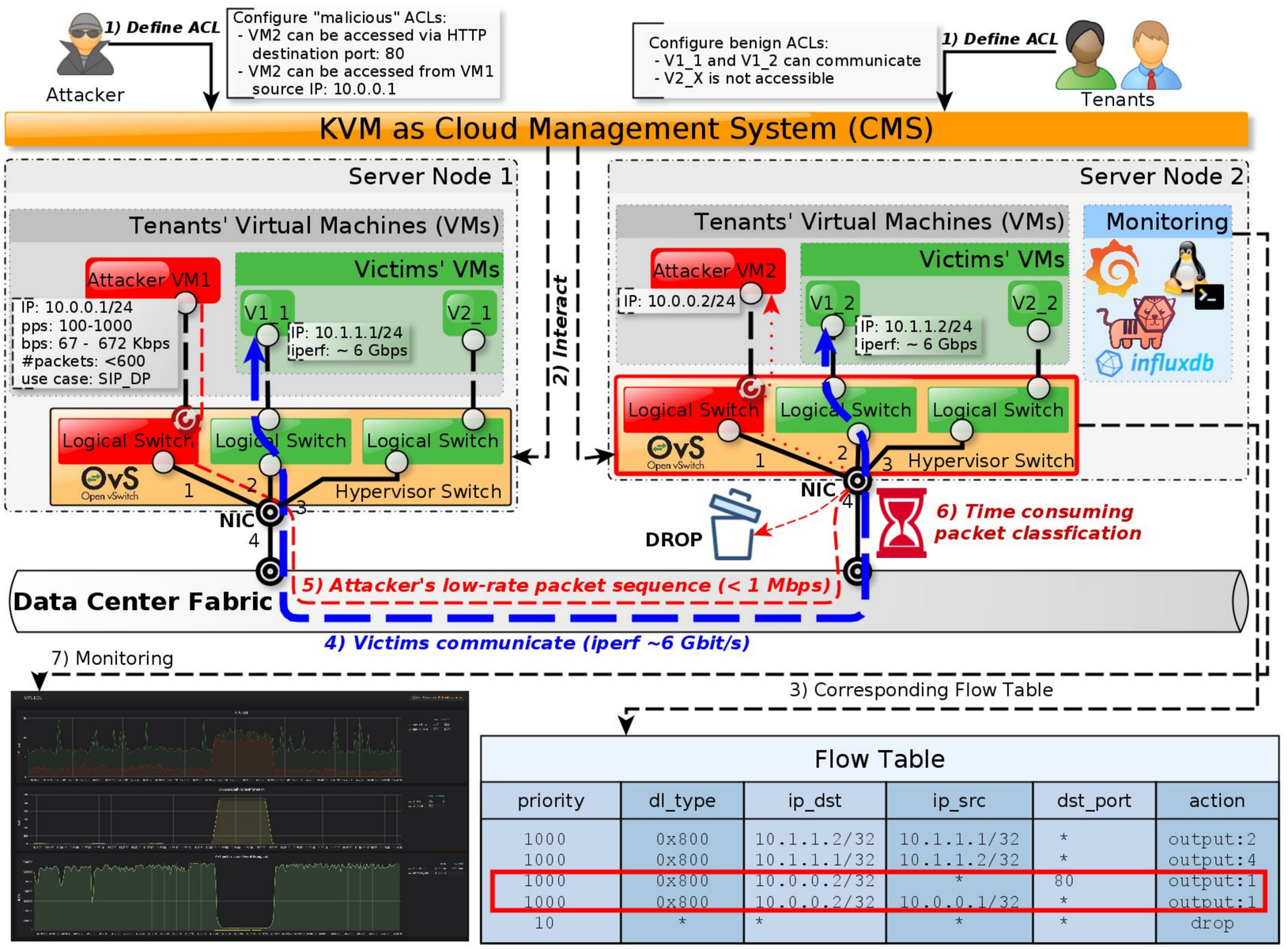
- Policy engine: a shared resource in the hypervisor switch that evaluates users' policies
- Injected policy: the attacker installs malicious state(s) into the Policy engine via CMS
- Malicious packet sequence: the attacker poses requests to the policy engine that are "difficult" to evaluate against the injected policy
- The hypervisor switch spends its time processing malicious requests: DoS for the rest of the tenants

Demo: DoS on Open vSwitch

- Policy engine: megaflow cache/tuple-space search
- Injected policy: security policy
- Malicious packet sequence: "port scan"
- Threat: network DoS for the rest of co-located tenants

Steps of the attack

1. Rent two VMs in the public cloud: the initiator and the receiver
2. Use the CMS APIs to set up a malicious network policy to filter the inbound/outbound traffic between your VMs
3. The corresponding ACL will be installed at the virtual ports connecting the VMs to the hypervisor switches
4. Initiates a "port scan" from the initiator VM to the receiver VM
5. The OVS switch instance at the receiver applies the injected ACL to the packets
6. Creates lots of megaflow cache entries/ masks in the OVS fast path classifier
7. Megaflow cache access is denied to victim VMs
8. Victim pushed out to the OVS slow path
9. Performance degradation/DoS



Flow Table

priority	dl_type	ip_dst	ip_src	dst_port	action
1000	0x800	10.1.1.2/32	10.1.1.1/32	*	output:2
1000	0x800	10.1.1.1/32	10.1.1.2/32	*	output:4
1000	0x800	10.0.0.2/32	*	80	output:1
1000	0x800	10.0.0.2/32	10.0.0.1/32	*	output:1
10	*	*	*	*	drop

Kubernetes network policy

```
apiVersion:
projectcalico.org/v3
kind: NetworkPolicy
metadata:
  name:malicious-policy
  namespace: default
spec:
  selector: role==’database’
  ingress:
  - action: Allow
    protocol: UDP
    source:
      nets:
      - 10.0.0.1
  - action: Allow
    protocol: UDP
    destination:
      ports:
      - 80
  - action: Allow
    protocol: UDP
    source:
      ports:
      - 12345
```

Effects

- An attacker can target particular cloud-based services using a combination of co-location & policy injection attack, or target all co-located tenants
- Reproduced in synthetic setups, OpenStack/OVN, and Kubernetes/OVN
- Default Kubernetes and OpenStack installs not affected
- Significant performance penalty

Mitigation

- Implement ACLs in iptables
- Switch the megaflow cache off
- Increase number of CPUs
- Offload the OVS datapath to SmartNIC
- Use jumbo frames to reduce per-packet load needed to be processed by OVS
- What's your idea?

UNICAMP INTRIG INFORMATION & NETWORKING TECHNOLOGIES RESEARCH & INNOVATION GROUP
NETLAB NETWORKED SYSTEMS RESEARCH LABORATORY University of Glasgow
universität wien
HSNLab SINCE 1992 MŰEGYETEM 1782