Policy Injection: A Cloud Dataplane DoS Attack

Levente Csikor
University of Campinas

Christian Rothenberg
University of Campinas

Dimitrios P. Pezaros
University of Glasgow

Stefan Schmid
University of Vienna

László Toka
Budapest Univ. of Techn. and Economics

Gábor Rétvári
Budapest Univ. of Techn. and Economics

ABSTRACT

Enterprises continue to migrate their services to the cloud on a massive scale, but the increasing attack surface has become a natural target for malevolent actors. We show *policy injection*, a novel algorithmic complexity attack that enables a tenant to add specially tailored ACLs into the data center fabric to mount a denial-of-service attack through exploiting the built-in security mechanisms of the cloud management systems (CMS) . Our insight is that certain ACLs, when fed with special covert packets by an attacker, may be very difficult to evaluate, leading to an exhaustion of cloud resources. We show how a tenant can inject seemingly harmless ACLs into the cloud data plane to abuse an algorithmic deficiency in the most popular cloud hypervisor switch, Open vSwitch, and reduce its effective peak performance by 80-90%, and, in certain cases, denying network access altogether.

1 INTRODUCTION

Enterprises increasingly offload their business-critical workloads to the public cloud to benefit from low infrastructure cost, flexible resource provisioning, etc. At the same time, they inevitably share the network infrastructure with other, potentially malevolent users. Hence, infrastructure security is a major concern for enterprises, also fueled by doubts on the adequacy of isolation between tenants' workloads in the shared data-center infrastructure. Like any complex system, the cloud data plane may also be susceptible to *algorithmic complexity attacks* [5], whereby an attacker exploits some intrinsic algorithmic deficiency by forcing the system spending its time processing malicious input, while denying access to benign users.

Cloud users can control communications permitted between their services by setting up appropriate ACLs in the hypervisor switches via the cloud management system (CMS). Here, we show that even the simplest Whitelist + Default-Deny type of ACLs a typical CMS would accept from users *may be a plausible target for algorithmic complexity attacks*, exploiting that *network packet filtering and classification is a computationally very difficult problem* [6]. We show in this demonstration that by injecting a specially crafted ACL into the CMS and feeding this ACL with a low-bandwidth (1–2 Mbps) covert packet stream, an attacker can bring down Open vSwitch (OVS), the most popular hypervisor switch through exhausting the flow caches that underlie the OVS fast-path packet classifier.

2 BACKGROUND AND ARCHITECTURE

Cloud Networking. The basic unit of operation users can deploy over the cloud is the `pod` or `VM`, a pack of software resources running in an isolated environment with limited access to the rest of the system. Network isolation is mostly driven by *microsegmentation*, a cloud security best-practice to protect pod-to-pod communication. Creating such isolation barriers within the workload is implemented by network policies in Kubernetes [1] or security groups in OpenStack [7], allowing users to impose L3/L4 forwarding policies. ACLs, used mainly in firewalls operate on the IP 5-tuple: the IP source and destination address, transport protocol and its ports, with any of the 5 fields potentially wildcarded.

The Open vSwitch pipeline. A *flow table* is an ordered set of wildcard rules operating over certain header fields, and a set of packet processing primitives (i.e., actions) to be applied to matching packets. For increased flexibility, OVS permits flow rules to overlap; if multiple rules in the flow table match, the one added first will be applied. However, this makes packet classification rather complex since, even in the case of non-overlapping flow rules, the complexity of any wildcard rule matching algorithm could be exponential [3].

To fasten packet classification, OVS relies on the fast-path/slow-path separation principle. The first packet of each flow is subjected to full flow-table processing on the slow path, and the flow-specific rules and actions are then cached in the fast path, which can then process the rest of the flow's packets efficiently. The fast path comprises two layers of *flow caches*: the *microflow cache* implements an exact-match store over all header fields; and the *megaflow cache (MF)* uses tuple-space search (TSS): entries matching on the same header fields are collected into a hash in which masked packet headers can be found fast. To reduce complexity, the slow path ensures that MF entries are non-overlapping, resulting in masks and associated hashes are searched sequentially until the first matching entry is found. This means that even if hash lookup is $O(1)$, the TSS algorithm still has to iterate through all hashes assigned to different masks, rendering TSS a costly linear search when there are lots of masks. In

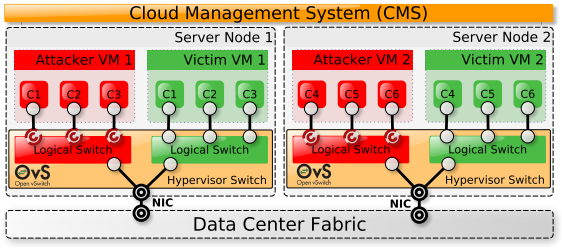

Figure 1: Architecture: The attacker installs ACLs at the virtual ports marked by red dots.

ip_src	action
00001010	allow
********	deny

(a) Binary ACL representation of the single-field network policy

Key	Mask	Action
00001010	11111111	allow
10000000	10000000	deny
01000000	11000000	deny
00100000	11100000	deny
00010000	11110000	deny
00000000	11111000	deny
00001100	11111100	deny
00001000	11111110	deny
00001011	11111111	deny

(b) Resultant non-overlapping MF entries

Figure 2: Simple ACL and the corresponding MF cache.

this note we show that this OVS flow cache infrastructure is inherently vulnerable to algorithmic-complexity attacks.

Architecture. To launch the attack as a user, we need the following ingredients: *(i)* the capability to define ACLs between our pods/VMs (this is provided by the CMS); *(ii)* a set of malicious ACLs; and *(iii)* an adversarial packet sequence, which will trash the MF with excess entries and masks, effectively sparking the DoS attack by triggering the above algorithmic deficiency (linear search) of the TSS scheme.

The test setup (see Fig. 1) comprises two server nodes, a data center fabric, and hypervisor switches (OVS in our case) providing network services to the pods/VMs provisioned at each server. First, the attacker injects an ACL to allow communication from `10.0.0.0/8` to her pods/VMs and deny everything else, resulting in the flow table shown in Fig. 2a. Note that there are different strategies to convert a flow table into a non-overlapping form: OVS in particular tries to wildcard as many bits as possible to get the broadest possible rules. This strategy results in exact-match entry for the allow-rule and 8 different key-mask pairs for testing the rest of the header bits (see Fig. 2b). We also need a packet sequence that will populate the MF with the "required" entries, but we omit the details in the interest of space. This technique creates 8 masks and so 8 iterations for executing the TSS. In the demo, we show how to extend this approach to launch an effective DoS attack.

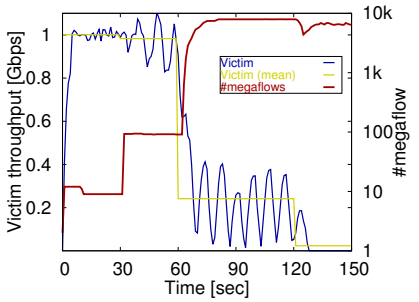

Figure 3: OVS degradation in Kubernetes: Attacker feeds her ACL with low-bandwidth packets at 60^{th} sec

During the demo, we will show how our technique can be applied to an arbitrary number of protocol fields, each resulting in a significant increase in the number of MF entries and masks in OVS. We will demonstrate that, by setting only 2 ACL rules matching solely on the IP source address and the L4 destination port (both ACLs are supported by Kubernetes/OpenStack), one can inject 512 MF masks/entries into the OVS fast path, slowing it down to 10% of the peak performance.

Then, we will show that, if the CMS allows us to also filter on the L4 source port (the Kubernetes networking plugin Calico does this), our attack technique can produce enough masks (8192) to a full-blown DoS attack (see Fig. 3). Attendees will also be engaged in discussions of, and be exposed to potential work-in-progress mitigation techniques and their trade-offs (e.g., joint troubleshooting techniques by tenants and provider [2], improved heuristics in OVS, flow cache-less softswitches [4]). Accompanying material can be found at *https://github.com/cslev/ovsdos*.

REFERENCES

[1] CNCF. 2018. Network Policies. https://kubernetes.io/docs/concepts/services-networking/network-policies. (2018).
[2] Levente Csikor et al. 2017. End-Host Driven Troubleshooting Architecture for Software-Defined Networking. In *IEEE Globecom 2017*. 1–7.
[3] P. Gupta and N. McKeown. 2001. Algorithms for Packet Classification. *Netwrk. Mag. of Global Internetwkg.* 15, 2 (2001), 24–32.
[4] László Molnár et al. 2016. Dataplane Specialization for High-performance OpenFlow Software Switching. In *SIGCOMM'16*. 539–552.
[5] Theofilos Petsios et al. 2017. SlowFuzz: Automated Domain-Independent Detection of Algorithmic Complexity Vulnerabilities. In *ACM CCS*. 2155–2168.
[6] Nick Shelly et al. 2014. Flow Caching for High Entropy Packet Fields. *SIGCOMM CCR* 44, 4 (2014).
[7] The OpenStack project. 2018. Manage project security. https://docs.openstack.org/nova/pike/admin/security-groups.html. (2018).

3 TECHNICAL REQUIREMENTS

3.1 Equipment to be used for the demo

A single laptop.

3.2 Space needed

Typical space provided by SIGCOMM - one large table

3.3 Setup time required

Up to 30 minutes.

3.4 Additional facilities needed, including power and any Internet access requirements

- Internet access (wired or wireless)
- Power source/extension cord
- One additional large screen for the showcase
- One poster stand for demonstrating the architecture