The Discrepancy of the MegaFlow Cache in OVS

Final Episode

Levente Csikor, National University of Singapore

Vipul Ujawane
IIT Kharagpur

Dinil Mon Divakaran
Trustwave (a Singtel Company)

Open vSwitch
December 8-9, 2020 | COVID-19 era, Online

The Discrepancy of the MegaFlow Cache in OVS
Final Episode

Levente Csikor, National University of Singapore

Vipul Ujawane
IIT Kharagpur

Dinil Mon Divakaran
Trustwave (a Singtel Company)
Recap: Flow caches, packet classification and TSS

- **Multi-layered cache architecture in the fast path**
  - Exact-match cache (EMC)
  - MegaFlow Cache (MFC)
    - arbitrary bitwise wildcards
- **Packet classification in the MFC**
  - Based on the Tuple Space Search (TSS) scheme
- **TSS in the MFC**
  - Entries matching on the same header bits are collected into tuples
    - Lookup in a tuple is fast
  - BUT: tuples are searched sequentially (until match found)
    - PKT_IN $\rightarrow$ APPLY_Mask $\rightarrow$ LookUp $\rightarrow$ Repeat until cache hit
  - if NO match:
    - Classify in the flow table
    - Cache the corresponding tuple in the MFC
Recap: Tuple Space Search

Can be a costly linear search in case of lots of masks!

dport=32777

<table>
<thead>
<tr>
<th>DST_PORT</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>output:1</td>
</tr>
<tr>
<td>*</td>
<td>drop</td>
</tr>
</tbody>
</table>

Flow Table

<table>
<thead>
<tr>
<th>DST_PORT</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>output:1</td>
</tr>
<tr>
<td>*</td>
<td>drop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dport=80</th>
</tr>
</thead>
<tbody>
<tr>
<td>80 allow</td>
</tr>
<tr>
<td>81 drop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dport=32768</th>
</tr>
</thead>
<tbody>
<tr>
<td>32768 drop</td>
</tr>
<tr>
<td>32769 drop</td>
</tr>
<tr>
<td>32770 drop</td>
</tr>
<tr>
<td>32771 drop</td>
</tr>
<tr>
<td>32772 drop</td>
</tr>
<tr>
<td>32773 drop</td>
</tr>
<tr>
<td>65535 drop</td>
</tr>
</tbody>
</table>
Discrepancy in the MFC

- For each flow table/ACL
  - Easy-to-craft packet sequence
  - Inflates the tuple space to a certain extent
  - Linear search process of TSS spends too much time on each packet
  - Overall packet processing speed drops down
  - Denial-of-Service

- Packet sequence characteristics
  - Legitimate
    - No explicit pattern
    - Cumbersome to detect and mitigate
  - Low-rate (< 1 Mbps)
    - Almost every packet spawns a new tuple
    - Exploits the 10 second expiration time in the MFC
Limitations of previous works

- **OVS and its kernel datapath**
  - When installed via the packet manager (e.g., `apt-get install openvswitch-switch`)
  - Kernel datapath is shipped by the Linux kernel
    - Different than the one of the OVS developers
    - No big fan of heavy caching -> no EMC

- **OVS-DPDK**
  - Same code base for the fast-path
    - MFC should work the same
Evaluations

- **Setup**
  - OVS in KVM (Xeon 6230, Mellanox CX-5)
  - iperf3 in the VMs for performance indicator
  - ~9000 tuples according to the ACL
  - Attack starts at the 20th second
  - Low rate: 1000 pps (~650 kbps, 64B)

- **OVS-kernel (2.10 - kernel 4.19.0-8- amd64)**
  - Good-old default setup

- **OVS-source (2.13.90 - manually compiled)**
  - Strange behavior: like EMC is being flushed after populated every time
  - Defeated at 4000 pps

- **OVS-DPDK (19.11)**
  - DPDK-accelerated OVS
  - Slightly worse base-line perf. due to iperf
  - Resurgence around the 45th second!
OVS-DPDK: Enhancements

- Ranking in the tuple space
  - 2016 patch
  - lib/dpif-netdev.c:
    - static void dpcls_sort_subtable_vector(struct dpcls cls)
    - Sort tuples in every second according their hit counts

- Result
  - Higher rate benign traffic can be found much faster
  - Malicious traffic requires more time to be classified, though!
  - Overall packet performance is still affected
OVS-DPDK: Defeating the ranking

- **Key aspect 1:** Linear search starts from the "end of the tuple space"

```c
/* pvector_insert(&my_pvector, &elem1, 1);
 * pvector_insert(&my_pvector, &elem2, 2);
 * ...
 * PVECTOR_FOR_EACH (iter, &my_pvector) {
 *     printf on *+iter*....
 *     ...
 *     elem2 to be seen before elem1...
 */
```

- **Key aspect 2:** Freshly inserted tuples are ranked the highest – inserted at the end

```c
static struct dpcls_subtable *
dpcls_create_subtable(struct dpcls *cls, const struct
netdev_flow_key *mask) {
...
/* Add the new subtable at the end of the pvector (with no
hits yet) */
pvector_insert(&cls->subtables, subtable, 0);
...)
```

- **Performance depends on the**
  - Rank of the benign flows
  - Number of masks in the MFC
  - Rate of the attack traffic
Tuple Space Explosion attack v2.0 (TSE 2.0)

- **Idea:** Keep the ranking process busy
- **How?**
  - Stop and restart attack
  - Let some "older" tuples expire (and therefore disappear)
  - Then, respawn them again
  - *Without increasing the attack rate*
- **Why?**
  - Malicious tuples will be ranked the highest again
    - Benign traffic will never be ranked high
  - We still maintain thousands of masks in the MFC
  - Attack rate is still low
    - Even lower due to the short pauses
    - 10 seconds attack time, 2 seconds pause
- **Result:** ranking defeated -> benign traffic can never resurge
TSE 2.0 does not work in multi-core setups
- Even against 2-core
- Spikes are the sleep times

What NOT to do:
- Simply increase attack rate
- Traffic trace will be looped faster
- No tuple will expire -> TSE 1.0

TSE 2.1: Idea
- Adjust the traffic trace
  - Send each packet $n$ times
- Increase the attack rate $n$-fold
- Tuples will expire and respawned
- Due to the attack rate:
  - Complete DoS
Discrepancy in the MFC is still there
- OVS-source with EMC behaves strange
  - Similarly to other unknown side-effects [1]
- OVS-DPDK with the ranking alleviates the issue
  - But we can overcome this by carefully adjusting the original attack vectors

More detailed study on arXiv

Levente Csikor
NUS-Singtel Cyber Security
Research & Development Laboratory
National University of Singapore