The Discrepancy of the MegaFlow Cache in OVS Part II.

Levente Csikor, Min Suk Kang, Dinil Mon Divakaran
National University of Singapore
Quick Recap from Part I.

- **Algorithmic deficiency in Tuple Space Search** scheme
  - used in the MegaFlow Cache (MFC)
- Easy to achieve
  - according to the flow table
    - as simple as “allow some but drop others”
  - less than 1 Mbps specially crafted packet sequence
  - Full Denial-of-Service (OVS performance drops close to 0%)
- Works in (public) cloud deployments
  - against co-located victims
  - No mitigation is available
    - low rate, no specific attack signature, completely legitimate packets
    - Kubernetes/OVN, OpenStack/Neutron/OVN, Docker/OVN, etc.

<table>
<thead>
<tr>
<th>src_IP</th>
<th>dst_port</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>80</td>
<td>allow</td>
</tr>
<tr>
<td>10.0.2.2</td>
<td>*</td>
<td>allow</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>drop</td>
</tr>
</tbody>
</table>
Recap: Packet Processing in OVS

- **Flow table**
  - ordered set of **wildcard rules**
  - operating on a **set of header fields**
  - **set of packet processing primitives**
  - *flow rules can overlap! (priorities)*

- **Fastening packet classification**
  - **First packet**
    - full-blown flow table processing
  - **Subsequent packets**
    - flow-specific rules and actions are **cached**

**Flow Table**

<table>
<thead>
<tr>
<th>ip_src</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0/8</td>
<td>output:1</td>
</tr>
<tr>
<td>*</td>
<td>drop</td>
</tr>
</tbody>
</table>

**ovs-vswitchd**

- slow path
- fast path
- user
- kernel

**OVS Kernel module**

**MegaFlow Cache – Tuple Space Search scheme**
Entries matching on the same headers are collected into a hash
- masked packet headers can be found fast
- However, masks and associated hashes are searched sequentially
- PKT_IN → APPLY_MASK → LookUp → Repeat until found

Can be a costly linear search in case of lots of masks!
Recap: Blow up the MegaFlow Cache

- **KEY FINDING**: More masks → slower packet processing
- For every *allow* rule
  - corresponding packet sequence to reach this end
- Strategy:
  - one packet for the *allow* rule
  - add a packet with each of the relevant bits inverted
    - 1 packet → 1 MFC mask

<table>
<thead>
<tr>
<th>Binary representation</th>
<th>DST_PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000 0101 0000</td>
<td>80 (allow rule)</td>
</tr>
<tr>
<td>0000 0000 0101 0001</td>
<td>81</td>
</tr>
<tr>
<td>0000 0000 0101 0010</td>
<td>82</td>
</tr>
<tr>
<td>0000 0000 0101 0100</td>
<td>84</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1000 0000 0101 0000</td>
<td>32848</td>
</tr>
</tbody>
</table>
Tuple Space Explosion (TSE) attack animated ;)

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>
Tuple Space Explosion

- Cache growth
  1) 16-bit DST_PORT -> 16 masks
  2) 32-bit SRC_IP -> 32 masks
- ONLY ONE allow rule on ONE HEADER FIELD
- Multiple allow rules on multiple header fields -> Exponential growth
- Matching on either 1) and 2) -> 512 masks

<table>
<thead>
<tr>
<th>Flow Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>DST_PORT</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>*</td>
</tr>
</tbody>
</table>
### Tuple Space Explosion (TSE) - IMPACT

- **FHO** – Full HW Offload
  - Mellanox ConnectX-4
- **GRO** – Generic Recv. Offload
  - *should be* enabled by default
- **UDP**: no offloading :( 
  - Dp (16 masks)
    - allow rule on `DST_PORT` only
  - SpDp (256 masks)
    - allow rules on `SRC_PORT` and `DST_PORT`
  - SipDp (512 masks)
    - allow rules in `SRC_IP` and `DST_PORT`
  - SipSpDp (8192 masks)
    - allow rules on `SRC_IP`, `SRC_PORT` and `DST_PORT`

![Graph showing the impact of MegaFlow Cache in OVS]

- **OVS 2.9.2 (stable) Ubuntu 18.04**
- **OVS+OVN’19, 10 Dec**
Tuple Space Explosion – Main takeaways

- Being aware/in control of the flow table
  - few thousandpps -> complete denial-of-service
- 10 sec timeout in the MFC
  - makes an adversary’s job easier
- Microflow cache might alleviate this, **BUT**
  - easily saturated in normal operation
  - or with high entropy in non-important headers in the attack sequence
    - e.g., TTL
  - disabled by default (OVS kernel module coming from the dist. repo)
Part II: In this talk

- **We DO NOT** present:
  - New deficiency of OVS/TSS
  - Implementation of another packet classifier
  - Improvement to the packet classifier itself

- **We DO** discuss:
  - Can the *attack* be *more generic without* the need of
    - *co-location*
    - and *flow table-awareness*
  - *Countermeasures*
Packet Classifier

<table>
<thead>
<tr>
<th>src_IP</th>
<th>dst_port</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.2.2</td>
<td></td>
<td>allow</td>
</tr>
<tr>
<td>*</td>
<td>993</td>
<td>allow</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>drop</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>drop</td>
</tr>
</tbody>
</table>

Virtualized Packet Classifier

Virtualized Packet Classifier

Virtualized Packet Classifier

Virtualized Packet Classifier

IDS

Storage

Social Media

eMail

Packet Classifier

Packet Classifier
Generic Tuple Space Explosion (TSE) Attack

- Challenge:
  - Blow up the MFC w/o knowing/in control of the flow table
- Possible?
  - How much effort does it need?
  - How successful can it be?
- Countermeasure?
How to generate the packets?

- **Being unaware of the flow table** -> **Difficult!**
  - All possible packets *could* work
    - $2^k$ packets for a header of $k$ bits
    - too much effort!
    - easily detectable e.g., portscan, volumetric (2.9 pbps in case of SipDp)

- Can’t we just use **random packets** instead?
Generic TSE Attack: Expectations

- What are the chances that a random packet spawns an MFC mask [1]?

- Key: number of wildcarded bits ($k$) for header length $h$

$$p_k(MFC) = \frac{2^k}{2^h}$$

- $1**** ***** ***** *****(32768) \sim 50\%$
- $0000000001** *****(64) \sim 0.1\%$

---

eneric TSE Attack: Results

- (M)easured and (E)xpected numbers for the different ACLs installed by a victim
  - Dp: DST_PORT only
  - SpDp: SRC_PORT + DST_PORT
  - SipDpSp: SRC_IP + DST_PORT + SRC_PORT
- 672 kbps (!) attack traffic
  - 90% performance drop
  - 1000 pps: 10Gbps -> 1 Gbps

![Graph showing the number of MFC masks against the number of packets]

- drop to 10%
Countermeasures

- **Detections hard**
  - legitimate traffic
  - no attack signature (full random packets)
  - low attack rate

- **MFCGuard (MFCg)**
  - Monitors the MFC
    - #masks > threshold
      - looks for TSE pattern
      - wipe out corresponding entries from the cache
  - **Attack traffic** goes to the **slow path** again
    - **benign traffic** remains (fast) in the fast path
MFCg

OVS+vswitch

fast path

slow path

OVS Kernel module

useful traffic

low-rate TSE attack traffic

MFCg
- Cleaned MFC -> normal throughput
- Neither documented nor expected behaviour
  - Attack traffic should be cached again
    - but they never will be
  - Constant overhead on the slow path
    - 1 kpps attack traffic = 15% overhead
    - 10 kpps attack traffic = 80% overhead
- GRO OFF
- Attack:
  - SipDp
  - 100 pps
More sophisticated algorithm is needed
- Wipe out only some select flows
- Maintain good balance between the fast path and slow path
- **Dynamically set a per-flow timeout in the MFC**
  - avoid uniform 10 sec timeout
  - more hits for a mask -> longer timeout
- **Prioritize**
  - Hashes with no masked bits (derived from flow table)
    - e.g., 80/ffff, 10.0.0.1/ffffffff
Conclusion

- **Tuple Space Search** algorithm has an **algorithmic-complexity vulnerability**
- Can be exploited by an adversary (easily)
- **Tuple Space Explosion attack**
  - against the infrastructure **via co-location**
    - full-blown denial-of-service
  - against an **arbitrary target**
    - substantial degradation-of-service
- **MFCguard**
  - keep the fast path clean for the benign traffic
Levente Csikor

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