Tuple Space Explosion: A Denial-of-Service Attack Against a Software Packet Classifier

Levente Csikor, Min Suk Kang, Dinil Mon Divakaran
Attila Kőrösi, Dávid Haja, Balázs Sonkoly, Dimitrios P. Pezaros, Stefan Schmid, Gábor Rétvári
Packet Classification in the Past

Packet Classifier

Facebook

eMail

Storage

IDS

Drop
Proliferation of virtualization

Packet Classifier

virtualized Packet classifier
(e.g., Open vSwitch, VPP)

Social Media
eMail

virtualized Packet classifier
(e.g., Open vSwitch, VPP)

Storage
IDS
Algorithmic packet classification is expensive on general purpose processors...

In this talk

- **Tuple Space Explosion (TSE):** *Family of novel Denial-of-Service (DoS) attacks* against the *de facto packet classifier* algorithm (Tuple Space Search scheme) used in Open vSwitch, VPP, GSwitch, etc.

- **Remote adversary** can degrade the performance to 12% of the baseline (10 Gbps) with only 672 kbps (!) attack traffic

- **Co-located adversary** can virtually bring down the performance to 0%

- Attack traffic is particularly hard to filter out:
  - *no attack signature* (packets w/ random headers)
  - *low-rate* (thousands of packets per second)
  - *legitimate packets*
Threat model

- **System model:**
  - typical multi-tenant cloud
  - OVS is used for packet processing
  - tenants use the Cloud Management System (CMS) to set up their ACLs to
    - access-control, redirect, log, etc.

- **Attacker’s goal**
  - send some packet towards the virtual switch that when subjected to the ACLs will exhaust resources

- **Attacker’s capability**
  - craft and send arbitrary packets to a target OVS
  - No privilege of the target (General TSE)
  - Co-locate with the target (Colocated TSE)
### Packet Classifier

**Social Media**
- src_IP: *
- dst_port: 993
- action: allow
- src_IP: *
- dst_port: 80
- action: allow

**Storage**
- src_IP: 10.0.2.2
- dst_port: *
- action: drop

**IDS**
- src_IP: *
- dst_port: *
- action: drop
Explosion in the Tuple Space

- **Problem**: more masks \(\rightarrow\) slower packet classification
- **Tuple Space Explosion phenomenon**:
  1) 16-bit TCP destination port \(\rightarrow\) 16 masks
  2) 32-bit source IP address \(\rightarrow\) 32 masks
  - And that’s only ONE *allow rule* on ONE *header*

- Multiple *allow rules* on multiple header fields result in an exponential growth \(\rightarrow\) cross-product
  - Matching on either 1) or 2) \(\rightarrow\) 16*32 = 512 *masks*
**Goal**: blow up the tuple space

- Spawn as many masks (and hashes) as possible
to make classification a costly linear search

- One packet for each bucket

- $\text{port}=[0, 64, 80, 81, \ldots, 32768]$ (16 packets)
Without the flow table → **Difficult**

- All possible packets seems fine
- **BUT:** $2^k$ packets for a header of $k$ bits!
  - too much effort
  - easily detectable (like a portscan, easily becomes volumetric)

**Can we just send** *random* *packets?*
**TSE w/ random packets**

- **Q:** What are the chances that a random header spawns a new mask (and hash)?

- *key finding is the number of wildcarded bits ($k$) for header length $h$*

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

- **32768/8000**

<table>
<thead>
<tr>
<th>32768</th>
<th>drop</th>
</tr>
</thead>
<tbody>
<tr>
<td>32769</td>
<td>drop</td>
</tr>
<tr>
<td>32770</td>
<td>drop</td>
</tr>
<tr>
<td>32771</td>
<td>drop</td>
</tr>
<tr>
<td>32772</td>
<td>drop</td>
</tr>
<tr>
<td>32773</td>
<td>drop</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>65535</td>
<td>drop</td>
</tr>
</tbody>
</table>

- **64/fff0**

<table>
<thead>
<tr>
<th>64</th>
<th>drop</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>drop</td>
</tr>
<tr>
<td>66</td>
<td>drop</td>
</tr>
<tr>
<td>67</td>
<td>drop</td>
</tr>
<tr>
<td>68</td>
<td>drop</td>
</tr>
<tr>
<td>69</td>
<td>drop</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>drop</td>
</tr>
</tbody>
</table>

- $1^{***} **** **** **** (32768) \sim 50\%$

- $0000 0000 01^{**} **** (64) \sim 0.1\%$

14/26
TSE w/ random packets

- (M)easured and (E)xpected numbers for different ACLs assumed to be installed by the victim drop to 10%.
  - Dp
  - dst_port only
  - SipDp
  - src_IP + dst_port
  - SpDp
  - src_port + dst_port
  - SipSpDp (full-blown)
  - src_IP + src_port + dst_port

15/26
Denial-of-Service

- Success rate of randomly generated packets
- 672 kbps (!) attack traffic → 88% performance drop
- 1,000 pps → reduce from 10 Gbps to 1,2 Gbps

What if the adversary has more knowledge/resources?
Co-located TSE attack

- Adversary leases resources in the cloud
- Configures its own ACL
- Sends only the required number of packets
  - one packet for each mask (and hash)

More significant service degradation – much less packets
- 1000 pps → thousands of masks → close to 0% (full DoS)

However:
- Attack is against the infrastructure not a specific target
- DoS against the co-located services “only”
Effects in a broader scale

- In a cloud, an attacker can easily exploit this!
- Several public cloud deployments are affected
  - Docker/OVN (based on OVS)
  - Kubernetes/OVN (based on OVS)
  - Contiv/VPP Kubernetes (based on VPP)
  - OpenStack/Neutron/OVN (based on OVS)
  - OpenStack/Neutron-VPP (based on VPP)
Countermeasures

- **Filtering out the attack traffic is hard**
  - legitimate traffic
  - no attack signature (random packets w/ random headers)
  - low-attack rate (thousands of packets per second)

- A long term solution
  - Different classifiers:
    - Hierarchical trees, HyperCuts, HaRP, etc.
MFC Guard (MFCg) in action
MFC Guard (MFCg)

- When MFC is cleaned the victim’s performance goes back to its baseline
  - attack packets → slow path
- CPU overhead?
  - 1 kpps attack rate = 15% CPU usage
  - 10 kpps attack rate = 80% CPU usage
General TSE

Random packets

Probability that from \( n \) random packets there will be at least 1 packet that sparks an MFC entry for a given \( k \) is:

\[
p_{(k,n)}(MFC) = (1 - (1 - p_k(MFC))^n) \times C_k
\]

\( C_k \) is the number entries for a given \( k \) (e.g., \( k=0, C_k = 2 \))

Expected value can be formalized by:

\[
\mathcal{E}_{(k,n)}(MFC) = \sum_{k=0}^{h} p_{(k,n)}(MFC)
\]
Countermeasures

- Immediate yet impractical remedies
  - offload ACL implementation to a different switch
    - others might suffer from the same attack
  - high performance gateway appliance
    - cannot help against an attack within the cloud
  - switch MFC completely OFF
    - biggest performance improvement so far
**Tuple Space Search**

- Entries matching on the same header are collected into a hash.
- Masked packet headers can be found fast.
- Masks and associated hashes are searched sequentially.
- Can be a costly linear search in case of lots of masks.

PKT_IN → APPLY_MASK → LookUp → Repeat until found

```plaintext
<table>
<thead>
<tr>
<th>TCP 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>dport=80</th>
<th>80/ffff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>drop</td>
</tr>
<tr>
<td>2</td>
<td>drop</td>
</tr>
<tr>
<td>3</td>
<td>drop</td>
</tr>
<tr>
<td>4</td>
<td>drop</td>
</tr>
<tr>
<td>5</td>
<td>drop</td>
</tr>
<tr>
<td>6</td>
<td>drop</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>63</td>
<td>drop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dport=32777</th>
</tr>
</thead>
<tbody>
<tr>
<td>64/fff0</td>
</tr>
<tr>
<td>65</td>
</tr>
<tr>
<td>66</td>
</tr>
<tr>
<td>67</td>
</tr>
<tr>
<td>68</td>
</tr>
<tr>
<td>69</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>79</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>0/ffc0</th>
<th>64/fff0</th>
<th>80/ffff</th>
<th>81/ffff</th>
<th>256/ff00</th>
<th>32768/8000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>drop</td>
<td>64</td>
<td>drop</td>
<td>80</td>
<td>allow</td>
</tr>
<tr>
<td>2</td>
<td>drop</td>
<td>65</td>
<td>drop</td>
<td>81</td>
<td>drop</td>
</tr>
<tr>
<td>3</td>
<td>drop</td>
<td>66</td>
<td>drop</td>
<td>80</td>
<td>allow</td>
</tr>
<tr>
<td>4</td>
<td>drop</td>
<td>67</td>
<td>drop</td>
<td>81</td>
<td>drop</td>
</tr>
<tr>
<td>5</td>
<td>drop</td>
<td>68</td>
<td>drop</td>
<td>80</td>
<td>allow</td>
</tr>
<tr>
<td>6</td>
<td>drop</td>
<td>69</td>
<td>drop</td>
<td>81</td>
<td>drop</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>63</td>
<td>drop</td>
<td>79</td>
<td>drop</td>
<td>80</td>
<td>allow</td>
</tr>
<tr>
<td>256</td>
<td>drop</td>
<td>257</td>
<td>drop</td>
<td>258</td>
<td>drop</td>
</tr>
<tr>
<td>259</td>
<td>drop</td>
<td>260</td>
<td>drop</td>
<td>261</td>
<td>drop</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>511</td>
<td>drop</td>
<td>65535</td>
<td>drop</td>
<td>32768</td>
<td>drop</td>
</tr>
<tr>
<td>32768</td>
<td>drop</td>
<td>32769</td>
<td>drop</td>
<td>32770</td>
<td>drop</td>
</tr>
<tr>
<td>32771</td>
<td>drop</td>
<td>32772</td>
<td>drop</td>
<td>32773</td>
<td>drop</td>
</tr>
<tr>
<td>32777</td>
<td>drop</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>65535</td>
<td>drop</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```