When Match Fields Do Not Need to Match: Buffered Packet Hijacking in SDN

Jiahao Cao, Renjie Xie, Kun Sun, Qi Li, Guofei Gu, and Mingwei Xu
Outline

• SDN Overview
• Background on SDN Rule Installation
• A New Vulnerability: Buffered Packet Hijacking
• Buffered Packet Hijacking Attacks
• Defense
• Conclusion
Outline

• SDN Overview
  • Background on Rule Installation
  • A New Vulnerability: Buffered Packet Hijacking
  • Buffered Packet Hijacking Attacks
  • Defense
  • Conclusion
SDN Overview

- **SDN applications (apps)**
  - Extend controller capacities and SDN functionalities

- **SDN controller**
  - Take centralized network control

- **SDN switches**
  - Forward and process flows according to the controller
Outline

- SDN Overview
- Background on SDN Rule Installation
- A New Vulnerability: Buffered Packet Hijacking
- Buffered Packet Hijacking Attacks
- Defense
- Conclusion
Rule Installation in SDN

• Packet-in
  • Query network decisions for a new flow
  • Contain a buffer ID and packet headers

• Flow-mod
  1. Install rules with match fields and actions
  2. Specify a buffer ID to release a buffered packet
Rule Conflict in SDN

• Conflict reason
  • **Multiple apps** process the same flow may generate conflicting rules

• Conflict abuse
  • Apps install conflicting rules to **override other apps’ decisions**
Rule Conflict Detection

- Rule conflict detection
  - Extract **match fields** and **actions** in all flow-mod messages
  - Check potential conflict when installing new rules

Do not consider potential **buffer ID abuse**

VerfiFlow (NSDI ’13), SE-Floodlight (NDSS ‘15), FortNOX (HotSDN ‘12)…
Outline

• SDN Overview
• Background on SDN Rule Installation
• A New Vulnerability: Buffered Packet Hijacking
• Buffered Packet Hijacking Attacks
• Defense
• Conclusion
Buffered Packet Hijacking Vulnerability

• Mechanism
  - Manipulate buffer IDs to hijack buffered packets

• Root Cause
  - No checking on the inconsistency between buffer IDs and match fields when installing rules
Outline

• SDN Overview
• Background on SDN Rule Installation
• A New Vulnerability: Buffered Packet Hijacking
• Buffered Packet Hijacking Attacks
• Defense
• Conclusion
Threat Model

• Attacker Objective
  • Exploit the vulnerability to attack all three SDN layers

• System Assumptions
  • SDN controllers, switches, and control channels are secure
  • Existing SDN defense may be deployed
  • Apps are untrusted, which may originate from third parties
  • A malicious app has basic permissions of listening packet-in and installing flow rules
Attacks and Testbed

• Four attacks
  • Attacking application
    1. cross-app poisoning
  • Attacking control plane
    2. control traffic amplification
  • Attacking data plane
    3. security policy bypass
    4. TCP connection disruption

• Real SDN testbed
  • Open source controller
    • Floodlight
  • Commercial SDN switches
    • EdgeCore AS4610-54T
  • Real background flows
    • Traffic trace from CAIDA
    • Crafted test flows
Attack 1: Cross-App Poisoning (CAP)

- A malicious app resends modified buffered packets to the controller

```
APP X
FLOW-MOD
buf_id: 1

h1
port1

S1

APP Y
PACKET-IN

PORT

h2
port2

APP X: FLOW-MOD
match: other flow
buf_id: 1
action: set-field (IP_SRC→IP_h2), output: controller

APP Y learns:
(Host, Port) = (h1, port1)

APP Y learns:
(Host, Port) = (h2, port1)

Incorrect mapping!
```

APP Y learns:
(Host, Port) = (h1, port1)

APP Y learns:
(Host, Port) = (h2, port1)
Evading Defense against CAP

- Existing CAP attacks and defense
  - Attack by modifying **shared data objects** in the **control plane**
  - Defend by checking information flow control policy violations*

- This CAP attack
  - Manipulate **buffered packets** in the **data plane**
  - Evade defense since there are **no policy violations**

* Ujcich, Benjamin E., et al. “Cross-app poisoning in software-defined networking.” CCS ’18
Attack 2: Control Traffic Amplification Bomb

- A malicious app copies massive buffered packets to trigger packet-in messages consuming bandwidth and computing resources.
Evading Defense against Packet-in Flooding

• Existing flooding attacks and defense
  • Attack by generating packets matching no rules to trigger massive packet-in messages
  • Detect malicious flows or adopt TCP SYN proxy to throttle TCP-based flooding*

• This flooding attack
  • Hijack buffered packets of benign flows to trigger massive packet-in messages
  • Generate no malicious flows and can hijack UDP flows

Attack 3: Network Security Policy Bypass

- A malicious app redirects buffered packets to different ports

Successfully bypass firewall!
Evading Defense against Security Bypass

• Existing security bypass attacks and defense
  • Generate **conflicting rules** to bypass security policies
  • Detect rule conflict to prevent security policy bypass*

• This attack
  • Manipulate **buffer IDs** to bypass security policies
  • Evade defense by generating **no conflicting rules**

Attack 4: TCP Connection Disruption

- TCP three-way handshake process
  - A TCP connection is established only after a successful TCP three-way handshake

The first packet of a TCP flow is always the TCP SYN packet.
Attack 4: TCP Connection Disruption

- A malicious app drops a buffered TCP SYN packet

Every 100 ms latency may cost 1% in business revenue for Amazon.
No existing SDN defense solutions consider this attack!
Hijacking Probability: **Intra-Chain** Hijacking

- **Single Processing Chain**
  - Apps in the same processing chain process packet-in and send flow-mod messages in turn

- **Success Condition**
  - A malicious app is **in front of** the app that will process the flow (target app)
Hijacking Probability: Inter-Chain Hijacking

• Multiple Processing Chains
  • Apps in different processing chains process packet-in and send flow-mod messages independently

• Success Condition
  • A malicious app could be in any position, if

\[
\sum_{i=1}^{\text{malicious}} \text{delay} < \sum_{i=1}^{\text{target}} \text{delay}
\]
Hijacking Probability: Experimental Results

- Experiments with two processing chains in real SDN testbed

- Intra-chain hijacking probability is either 0 or 100%
- Inter-chain hijacking probability decreases when the malicious app moves towards tail, e.g., from 100% to 36.3% for Load Balancer
Hijacking Probability: Theory Analysis

- Derive hijacking probability from processing chain model

- $a_{r,c}$: malicious app, the $c$-th application in the $r$-th processing chain
- $a_{i,j}$: target app, the $i$-th application in the $j$-th processing chain
- $f_{i,j}$: probability density function of processing delays in $a_{i,j}$

- Intra-chain hijacking probability:
  $$p_{\text{intra}}(a_{r,c}, a_{r,j}) = \begin{cases} 100\%, & \text{if } j \in \{1, 2, \ldots, c-1\} \\ 0, & \text{if } j \in \{c+1, c+2, \ldots, n_r\} \end{cases}$$

- Inter-chain hijacking probability:
  $$p_{\text{inter}}(a_{r,c}, a_{i,j}) = \prod_{k=1}^{j} f_{i,k}(t_k - t_{k-1}) \prod_{k=1}^{j+c-1} f_{r,k}(t_{j+k-1} - t_{j+k}) \cdot \prod_{k=1}^{j+c-1} dt_k \cdot dz$$

Details in our paper!
Outline

• SDN Overview
• Background on Rule Installation
• A New Vulnerability: Buffered Packet Hijacking
• Buffered Packet Hijacking Attacks
• Defense
• Conclusion
Defense: ConCheck

- Add **consistency check** between buffer IDs and match fields

Detection Example

ConCheck Architecture
Outline

- SDN Overview
- Background on Rule Installation
- A New Vulnerability: Buffered Packet Hijacking
- Buffered Packet Hijacking Attacks
- Defense
- Conclusion
Conclusion

• We discover a new vulnerability in SDN rule installation.

• We identify four buffered packet hijacking attacks that disrupt all SDN layers and can evade all existing defense systems.

• We propose a lightweight and application-transparent countermeasure.
Thank you!

Kun Sun
ksun3@gmu.edu
Backup: Permissions

- The ratio of applications with the permission of listening packet-in messages and installing flow rules

<table>
<thead>
<tr>
<th>Controller</th>
<th>Total APPs</th>
<th>APPs with the Permission</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenDaylight Neon†</td>
<td>13</td>
<td>6</td>
<td>46.2%</td>
</tr>
<tr>
<td>ONOS v2.1.0-rc1</td>
<td>97</td>
<td>23</td>
<td>23.7%</td>
</tr>
<tr>
<td>Floodlight v1.2</td>
<td>29</td>
<td>12</td>
<td>41.4%</td>
</tr>
<tr>
<td>RYU v4.31</td>
<td>28</td>
<td>19</td>
<td>67.9%</td>
</tr>
<tr>
<td>POX eel version</td>
<td>18</td>
<td>11</td>
<td>61.1%</td>
</tr>
</tbody>
</table>

† Only counting the applications implemented with openflowplugin.

Many apps have the permissions
Mainstream SDN vendor Pica8

- Acknowledged our report and said "we have filed tracking tickets and are waiting for product management decision on releasing the fix in major/minor or patch builds"

Mainstream carrier-grade SDN controller ONOS

- Helped us file a defect in the ONOS community with the comment that "the defect will be visible to the community and this info can be available for someone to pick it up to fix it"

Popular SDN controller RYU

- Several developers and users in the community confirmed our report
Evaluation on ConCheck

- We implement a prototype of ConCheck in Floodlight

![CDF graph with Flow Setup Time (ms) on the x-axis and CDF on the y-axis. The graph compares Baseline and ConCheck, showing minor overhead for apps to install flow rules.]