Tackling Control Plane DoS Attacks in Software-Defined Networking

Fabio De Gaspari
Sapienza University, Computer Science Dept.
degaspari@di.uniroma1.it

Joint work with:
M. Ambrosin, M. Conti
University of Padua, Mathematics Dept.
R. Poovendran
University of Washington, Electrical Engineering Dept.
Outline

- SDN Overview
  - OpenFlow

- DoS attacks in SDN
  - Control plane saturation attack

- State-of-the-art (AvantGuard) and its limitations

- Our solution: LineSwitch

- Evaluation

- Conclusion
Intro: Software Defined Networking

- Logic/infrastructure decoupling
  - Reduces switch complexity while enhancing flexibility

- Control plane as middleware
  - Exposes set of open APIs to applications

- Switches programmable through standard interface
  - OpenFlow is the most widely accepted switch-related standard
Introduction: OpenFlow

- Switch maintains a set of Flow Tables
  - Organised in a pipeline
- Each flow table hosts a set of Flow Rules
  - Flow rules define what actions to perform on a given network flow
- Control plane installs flow rules
  - Either statically or dynamically
Control Plane Saturation Attack

- Exploits extensive communication between data and control plane
- Attacker floods an OF-switch with unique network flows
- For each flow, the OF-switch contacts the controller to ask for a flow rule
- Controller needs to analyse requests, define actions and send back a response
- With a high enough flood rate, the controller can be overwhelmed
AvantGuard is a switch-level SDN extension which aims at defeating SYN-based flooding.

- Applies a set of well-known SYN-flooding mitigation techniques to SDN:
  - Syn Cookies
  - Syn Proxy

- AG is composed of two distinct modules:
  - Connection Migration module
  - Actuating Trigger module
AvantGuard 2/3: Connection Migration

- Switch acts as a SYN Proxy
- Switch contacts controller *iff* handshake with A is completed
- Upon receiving permission, switch completes connection with Host B
- Switch enters relay phase
AvantGuard 3/3: Limitations

- The switch can not know Host B's ISN during the TCP handshake with Host A
  - For each packet, it is forced to translate sequence and ACK number

- Connecting to Host B, the switch can not use Host A's IP
  - For each packet, the switch needs to translate IP addresses
  - Needs to use different ports to migrate connections to an \(<IP, port>\) pair

- Therefore, the switch:
  - Is forced to maintain state \(for\ each\ connection\)
  - Can not migrate more connections than the available port numbers
Buffer Saturation Attack

- Exploits the need for a translation table for each connection
- The attacker generates a high number of complete TCP handshakes
- The switch will create an entry in the translation table for each one of them
- When the table is saturated, the switch will not be able to migrate any legit connection
- Experimental data shows buffers can be saturated in under 100 seconds with a modest attack rate of 1Mbps
Breaking the end-to-end Semantics

- Spoofed lower layer information can break higher level applications

- Proxying prevents proper setup of TCP options
  - None of the options for high performance can be used
  - MSS must be set to a conservative low value, increasing fragmentation

- A change in the traffic path from the switch will destroy the TCP connection
  - Since the switch uses its IP address for the migration, the destination host has no knowledge of the real client.
Our Solution: LineSwitch 1/2

- LineSwitch (LS) combines probabilistic proxying and blacklisting
- High resilience to buffer saturation attack
  - Configurable through proxying probability parameter $P_p$
- Minimal use of proxying preserves TCP semantics
- Exponentially increasing blacklist duration for attackers
- Reduced overhead: LS is on average 30% faster than AG
Our Solution: LineSwitch 2/2

1. **Incoming SYN packet from IP_A**
   - First connection from IP_A
     - Use SYN proxy
   - Select a random \( n \in [0,1] \)
     - \( n \leq P_p \)?
       - Yes
         - Blacklist IP_A for \( T \times 2^{\text{count}_A} \) seconds
       - No
         - Enter relay phase
     - No
       - Use normal OpenFlow pipeline
Evaluation 1/3: System Model

- Attacker, client and server connected to single OF-switch
- Evaluated web page retrieval times under different attack rates
- Evaluated resilience to buffer saturation attack, with varying attack rates and buffer sizes
Evaluation 2/3: Web Page Retrieval

(a) Success rate
(b) Average retrieval time

Average retrieval time and success rate under different attack rates
Evaluation 3/3: Buffer Saturation Attack

(b) buffer size $2^{20}$Bytes

(c) buffer size $2^{22}$Bytes

Time required to saturate translation table with varying size and attack rate
Conclusion

• Extensive communication between data and control plane opens exploitable vulnerabilities in SDN

• Current state-of-the-art solution suffers from major design flows

• LineSwitch offers the same level of protection as AvantGuard, whilst reducing overhead by ~30% and protecting against buffer saturation attacks

• SDN is a promising alternative to current network architecture, but needs thorough security evaluation by specialists before adoption.
End Presentation

Thank you

Q&A
Offerta formativa del Dipartimento di Informatica

LA SCADENZA DELLA DOMANDA E’ PREVISTA PER IL 10 DICEMBRE 2015
Grazie

Per maggiori informazioni

Indirizzi e-mail
mastersicurezza@di.uniroma1.it
mastergovernance@di.uniroma1.it

Sito Web
http://mastersicurezza.uniroma1.it/

Oppure contattateci al box
## Appendix: Experimental Data

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Avg. Time</th>
<th>Std. Dev.</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenFlow</td>
<td>0.404 s</td>
<td>0.001 s</td>
<td>N.A.</td>
</tr>
<tr>
<td>AVANT-GUARD</td>
<td>0.573 s</td>
<td>0.014 s</td>
<td>41.83%</td>
</tr>
<tr>
<td>LineSwitch</td>
<td>0.435 s</td>
<td>0.030 s</td>
<td>7.67%</td>
</tr>
</tbody>
</table>

Page retrieval with no flooding

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Avg. Time</th>
<th>Success Rate</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenFlow</td>
<td>2.406 s</td>
<td>100%</td>
<td>495.54%</td>
</tr>
<tr>
<td>AVANT-GUARD</td>
<td>0.565 s</td>
<td>100%</td>
<td>39.85%</td>
</tr>
<tr>
<td>LineSwitch</td>
<td>0.411 s</td>
<td>100%</td>
<td>1.73%</td>
</tr>
</tbody>
</table>

Page retrieval with 3Mbps flooding

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Avg. Time</th>
<th>Success Rate</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenFlow</td>
<td>∞</td>
<td>0%</td>
<td>∞</td>
</tr>
<tr>
<td>AVANT-GUARD</td>
<td>0.568 s</td>
<td>100%</td>
<td>36.92%</td>
</tr>
<tr>
<td>LineSwitch</td>
<td>0.426 s</td>
<td>100%</td>
<td>5.45%</td>
</tr>
</tbody>
</table>

Page retrieval with 6.5Mbps flooding
LineSwitch Flow

- First connection flag is cleared after a configurable time $\delta$ of inactivity
- $P_p$ value of 0 degenerates in the standard OF pipeline, removing proxying benefits.
- $P_p$ value of 1 degenerates in proxying every packet, exposing the switch to buffer saturation attacks.

**Flowchart:**
- **Incoming SYN packet from $IP_A$**
  - First connection from $IP_A$
    - **Select a random $n \in [0,1]$**
      - $n \leq P_p$?
        - Yes: Use SYN proxy
          - Is handshake complete?
            - Yes: Use normal OpenFlow pipeline
              - No: Enter relay phase
        - No: count_IP_A = count_IP_A + 1
          - Blacklist IP_A for $T \times 2^{count_IP_A}$ seconds
            - $n \leq P_p$?
              - Yes: Use SYN proxy
                - Is handshake complete?
                  - Yes: Use normal OpenFlow pipeline
                    - No: Enter relay phase
              - No: Blacklist IP_A for $T \times 2^{count_IP_A}$ seconds
                - $n \leq P_p$?
                  - Yes: Use SYN proxy
                    - Is handshake complete?
                      - Yes: Use normal OpenFlow pipeline
                        - No: Enter relay phase
                  - No: Blacklist IP_A for $T \times 2^{count_IP_A}$ seconds