Securing Network Application Deployment in Software Defined Networking

Yuchia Tseng, Farid Naït-Abdesselam, and Ashfaq Khokhar
Outline

- Introduction to OpenFlow-based SDN
- Security issues of network application deployment in SDN
- Securing network application deployment in SDN
- Prototype evaluation
- Conclusion
**Overview of OpenFlow-based SDN**

**Software-defined Networking (SDN)**
- A new network paradigm decouples the control plane from the data plane
- Benefit: Provide centralized control and visibility over network
- Architecture
  - 3 planes: Data plane, Control plane, & Application plane
  - 3 interfaces: Southbound, Northbound, & Eastern/Western bound interfaces

Diagram:
- **Northbound interface** - OSGi, RESTful
- **Southbound Interface** - OpenFlow etc
- **Data plane** - network devices
- **Control plane** - SDN controller
- **Application plane** - network applications
- **Eastern/Western interfaces**
Overview of OpenFlow-based SDN

- **Data plane**
  - Networking devices for forwarding packets
  - Example: Open vSwitch, HP 2920/3500 Switch Series etc

- **Control plane**
  - Network OS, or called SDN controller, running on general purpose HW & OS
  - Example: OpenDaylight, ONOS, Floodlight, Ryu, NOX etc

- **Application plane**
  - Networking applications installed on SDN controller to enable network becoming intelligent
  - Example: Firewall, Load balancer, and IDS etc

- **Southbound interface**
  - Communication protocol between data plane & control plane
  - Example: OpenFlow, NETCONF, Border Gateway Protocol (BGP), Open vSwitch Database Management Protocol (OVSDB), MPLS Transport Profile (MPLS-TP) etc

- **Northbound interface**
  - Application programming interfaces (APIs) between control plane & application plane
  - Example: OSGi, native Java/Python APIs, RESTful APIs etc

- **Eastern/western interfaces**
Controller uses OpenFlow to keep listening what happens on the data plane, and then sends the network events to the application plane

**OpenFlow messages**
- packet_in: send a captured packet to the controller (e.g., a miss in the match tables)
- packet_out: inject packets into the data plane
- flow_mod: modify the state of an OpenFlow switch (e.g., add a flow entry)
- port_mod: modify the state of an OpenFlow port
- port_status: indicate a change of port status
• If controller uses OpenFlow protocol to keep listening what happens on the data plane

• How can the network applications **KEEP LISTENING** the network events on the data plane through the SDN controller?
  - Internal APIs, such as OSGi, Java/Python APIs
  - External APIs, such as RESTful APIs(JSON-RPC)
Security Issues of Network Application Deployment in SDN

- **Security problem of using internal APIs**
  - Easier to be compromised by code injection

Floodlight crash by calling `System.exit()`

Floodlight memory leakage by inserting data into list

More problems: run infinite loop, create numerous threads, listen to the network traffic, or insert code through JNI etc

Security Issues of Network Application Deployment in SDN

- **Security problem of using external APIs**
  - Can be compromised by API abuse
  - **READ permission**: Saturating the bandwidth of the northbound interface by using infinite loop to request the northbound APIs
  - **ADD permission**: Flow tables is limited. Eg., High-performance chips EZchip NP-4 stores 125 000 – 1 000 000 flow entries. The attacker can flush out the high priority flow rules by the low priority ones
  - **UPDATE & REMOVE permission**: The APIs can be used to compromise the higher priority flow rule due to the coarse-grained access control

<table>
<thead>
<tr>
<th></th>
<th>FLOODLIGHT</th>
<th>ODL</th>
<th>ONOS</th>
<th>RYU</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAED</td>
<td>Packet dropping</td>
<td>Responding slower</td>
<td>Packet dropping</td>
<td>Responding slower</td>
</tr>
<tr>
<td>ADD</td>
<td>Flow entry limitation: 148223</td>
<td>Flow entry limitation: 140000</td>
<td>Flow entry limitation: 45000</td>
<td></td>
</tr>
<tr>
<td>UPDATE</td>
<td>../sal-flow:update-flow</td>
<td>../devices/&lt;deviceld&gt;</td>
<td>../flowentry/modify</td>
<td></td>
</tr>
<tr>
<td>REMOVE</td>
<td>../sal-flow:remove-flow</td>
<td>../flows/&lt;deviceld&gt;/&lt;flowId&gt;</td>
<td>../flowentry/clear/&lt;dpid&gt;</td>
<td></td>
</tr>
</tbody>
</table>

So, how to deploy network application securely for SDN controller?

- Internal APIs (OSGi, Java/Python APIs etc)?
  - Malicious code injection...
- External APIs (JSON-RPC)?
  - API abuse
  - Hard to get the network events in real time
- IPC?
  - System-level command injection, eg:
    - `Runtime.getRuntime().exec("shutdown -s -t 0");`
  - High complexity to deploy on the current existing controllers
So, how to deploy network application securely for SDN controller?

- Internal APIs (OSGi, Java/Python APIs etc)?
  - Malicious code injection...
- External APIs (JSON-RPC)?
  - API abuse
  - Hard to get the network events in real time
- IPC?
  - System-level command injection, eg:
    - `Runtime.getRuntime().exec("shutdown -s -t 0");`
  - High complexity to deploy on the current existing controllers
**Application plane controller**
- Dedicate all the requests from the application plane
- Application authentication
- Application authorization (access control)
- Application resource isolation, control, & monitoring
- Message-driven service (instead of server-client mode)

**Data plane controller**
- Interpret the network rules ↔ OF entries
- Communicate with the data plane

Securing Network Application Deployment in SDN
Securing Network Application Deployment in SDN

- **Policy Engine**: define high-level policy
- **Parser**: translate high-level policies to Access Controller & Resource Controller
- **Resource Controller**: control and monitor resource usage of network application
- **Application Sandbox**: isolate the resource usage of the network application
- **Authentication module**: authenticate the network application
- **Access Controller**: authorize the requests of network applications
- **Message Broker**: provide message-driven service to network applications
Securing Network Application Deployment in SDN

- **Policy Engine**: define high-level policy
- **Parser**: translate high-level policies to Access Controller & Resource Controller
- **Resource Controller**: control and monitor resource usage of network application
- **Application Sandbox**: isolate the resource usage of the network application
- **Authentication module**: authenticate the network application
- **Access Controller**: authorize the requests of network applications
- **Message Broker**: provide message-driven service to network applications
Securing Network Application Deployment in SDN

- **Policy Engine**: define high-level policy
- **Parser**: translate high-level policies to Access Controller & Resource Controller
- **Resource Controller**: control and monitor resource usage of network application
- **Application Sandbox**: isolate the resource usage of the network application
- **Authentication module**: authenticate the network application
- **Access Controller**: authorize the requests of network applications
- **Message Broker**: provide message-driven service to network applications
• Policy Engine: define high-level policy
• Parser: translate high-level policies to Access Controller & Resource Controller
• **Resource Controller**: control and monitor resource usage of network application
• Application Sandbox: isolate the resource usage of the network application
• Authentication module: authenticate the network application
• Access Controller: authorize the requests of network applications
• Message Broker: provide message-driven service to network applications

Securing Network Application Deployment in SDN
Securing Network Application Deployment in SDN

- Policy Engine: define high-level policy
- Parser: translate high-level policies to Access Controller & Resource Controller
- Resource Controller: control and monitor resource usage of network application
- **Application Sandbox**: isolate the resource usage of the network application
- Authentication module: authenticate the network application
- Access Controller: authorize the requests of network applications
- Message Broker: provide message-driven service to network applications
Securing Network Application Deployment in SDN

- **Policy Engine**: define high-level policy
- **Parser**: translate high-level policies to Access Controller & Resource Controller
- **Resource Controller**: control and monitor resource usage of network application
- **Application Sandbox**: isolate the resource usage of the network application
- **Authentication module**: authenticate the network application
- **Access Controller**: authorize the requests of network applications
- **Message Broker**: provide message-driven service to network applications
- Policy Engine: define high-level policy
- Parser: translate high-level policies to Access Controller & Resource Controller
- Resource Controller: control and monitor resource usage of network application
- Application Sandbox: isolate the resource usage of the network application
- Authentication module: authenticate the network application
- Access Controller: authorize the requests of network applications
- Message Broker: provide message-driven service to network applications
High-level policy in YAML

Json for Marathon framework

ACLs for Kafka server
Access Controller
- App 1 can read DeviceInfo messages
- App 1 is rejected to read Configuration messages

Resource Controller
- An application contains a malicious loop to keep inserting data in a HashMap

When the code runs on the runtime of controller (JVM for Floodlight), after 1 minute, the controller can no longer serve the data plane.

When the code runs in the sandbox (Docker with 0.2 CPU & 128 RAM), the controller keeps serving the data plane.
Resource usage monitoring dashboard (implemented on Floodlight)

- System-level resource usage
- JVM-level resource usage
- Application sandbox resource usage
**Prototype Evaluation**

**Testbed configuration**

<table>
<thead>
<tr>
<th>Component</th>
<th>Tool</th>
<th>Sandbox number</th>
<th>Processing time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service broker</td>
<td>Apache Kafka</td>
<td>1</td>
<td>2.5946</td>
</tr>
<tr>
<td>Resource controller</td>
<td>Apache Mesos API and Sigar</td>
<td>2</td>
<td>2.9099</td>
</tr>
<tr>
<td>Application deployer</td>
<td>Marathon framework</td>
<td>3</td>
<td>3.3204</td>
</tr>
<tr>
<td>Application sandbox</td>
<td>Docker</td>
<td>4</td>
<td>4.4183</td>
</tr>
<tr>
<td>Policy</td>
<td>YAML</td>
<td>5</td>
<td>6.8173</td>
</tr>
<tr>
<td>Parser</td>
<td>Java application</td>
<td>6</td>
<td>7.3950</td>
</tr>
<tr>
<td>SDN controller</td>
<td>Floodlight master (v1.2)</td>
<td>7</td>
<td>9.8372</td>
</tr>
<tr>
<td>Network simulator</td>
<td>mininet</td>
<td>8</td>
<td>8.9402</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 16.04.2</td>
<td>9</td>
<td>9.5253</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel i7</td>
<td>10</td>
<td>13.2072</td>
</tr>
<tr>
<td>Memory</td>
<td>16G</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The average processing time for delivering 10 thousands *packet_in* messages from the data plane controller to 1-10 network application(s)

Processing time for delivering 20 thousands *packet_in* messages to 1-10 network applications
Conclusion

Problems
• How to deploy network applications securely on the SDN controller?

Solution
• Functional split of the SDN controller & security-by-design
• Application controller:
  ➔ Resource Controller: resource isolation, control, and monitoring → against code injection and command injection
  ➔ Message Broker + Access Controller → against API abuse

Preliminary results
• This architecture can implement in the existing SDN controller by adding a new module (agent)
• The latencies keeps around 5 milliseconds in long term for delivering to 10 network applications
Thank you!
Questions?

www.irt-systemx.fr