Retaining Control Over SDN Network Services

Christian Röpke and Thorsten Holz

International Conference on Networked Systems (NetSys '15)
Software-Defined Networking

• What's new?
  - Control programs are decoupled from the network devices
  - Network devices are programmable via an open interface
    → Flexibility and cost-efficiency

• Benefit: we can program our networks as needed.
Flexible SDN Controllers

- Network service installation on static SDN controllers
  - Compile, stop, replace, start
- Runtime flexibility

First SDN App Store by HP

SDN App Store
Powered by HP

Network Services
Network Services

• Examples
  - Topology manager
  - Learning switch
  - Firewall
  - Load balancer

• Typical functions
  - Abstraction of details
  - Counterpart to business applications

• Load balancer example
  - Create pool, add virtual IP, add pool members
Network Services

- **Examples**
  - Topology manager
  - Learning switch
  - Firewall
  - Load balancer

- **Typical functions**
  - Abstracting details
  - Counterpart to business applications

Network services build an important part of the SDN controller's functionality.

Installation at runtime enables appropriate flexibility in operating real networks.

There will hopefully arise a huge market of 3rd-party network services.
Threat Model

- Distribution of *malicious* network service
  - Fool network operator
  - Trigger vulnerabilities in validation functions
  - Obfuscate malicious logic or steal signing key
  - Launch targeted attack
Research Questions

• How will SDN controllers react on malicious network services?
• How big will the impact be?
• Do current SDN controllers provide any protection against such attacks?
• If not, how can we improve the security of SDN controllers?
Security Analysis

- Examined SDN controllers
  - Beacon
  - Floodlight
  - OpenDaylight
  - HP SDN VAN controller

- Test setup
Test Results

- Malicious network services
  - Controller shutdown
  - Malware execution and malicious function loading
  - Remote access to controller
- Found protection mechanism
  - Monitoring process restarts controller process (HP)
    → Easy to bypass
- Totally lost control for controllers running as root
Lessons Learned

All current SDN controllers share a critical problem with respect to malicious network services.

Root cause:
SDN controllers have a lack of capabilities to retain control over network services.
How Can We Retain Control?

- How can we harden SDN controllers and provide protection against malicious logic?

**Idea**

- Contain each network service in a sandbox
- Control system-level operations
- In case of critical / sensitive operations
  - Consult access control database
  - Grant or deny access
Architecture Overview
Capabilities

• Modes
  – Detection
  – Protection

• Easy to use northbound interface
  – Start, stop, status
  – Add, remove, list, …

• Automated transformation of log entries into sandbox configurations
Detection and Protection Mode

Log File

```
... WARN 'net-service-xy' needs
     ('java.net.SocketPermission" "1.2.3.4" "connect,resolve")
... WARN 'net-service-xy' needs
     ('java.io.FilePermission" "/tmp/file" "write")
... WARN 'net-service-xy' needs
     ('java.io.FilePermission" "/tmp/file" "execute")
... java.security.AccessControlException: access denied
     ('java.lang.RuntimePermission" "exitVM.0")
... at org.opendaylight.controller.secapp...checkPermission(...)
... at org.opendaylight.controller.shutdown.Shutdown.init(...)
```

Network Service

... sensitive operation
operation
... sensitive operation
... sensitive operation
...
Gratis Security
Performance

- Controller throughput test with cbench
- Low performance overhead
- Variation caused by controller
Challenge

- *Benign* network services most likely depend on sensitive operations (e.g., use existing service, register as new service)
- Needed permissions for correct operation are typically unknown beforehand
- But proper operation requires correct sandbox configuration
- Detection mode helps but may not reveal all sensitive operations
Ease Sandbox Configuration

• Idea
  – Developer can specify needed permissions
  – Operator can review and decide on permission requests
  – SDN controller blocks network service installation until all permission requests are processed
  – Operator can revise the initial sandbox configuration
Implementation

- Adapt install, update and remove procedures
  - Install / update
    - Permission file is taken into account
    - Each permission request is displayed
    - Controller waits until each request is processed
  - Remove
    - Look for *uninstalled* events
    - Remove according policy rules
- Protect adapted procedures against modification
Evaluation

• Effectiveness
  – No permission file
  – Permission file
  – Functioning tests while console is blocked

• Performance
  – Considerable overhead during installing and updating network services
    → Mainly caused by the time operators need to review and decide on displayed requests
Conclusion

- Current SDN controllers share a critical problem with respect to malicious network services.
- Our containment mechanism allows to retain control over network services by means of sandboxing.
- Additionally, we provide a way to ease sandbox configuration for better usability.
- In summary, we could improve SDN controller security without a significant reduction in performance.
Questions?