OpenFlow: A Security Analysis

Rowan Klöti\textsuperscript{1}  Vasileios Kotronis\textsuperscript{2}  Paul Smith\textsuperscript{3}

\textsuperscript{1}rkloeti@alumni.ethz.ch
ETH Zurich

\textsuperscript{2}vkotroni@tik.ee.ethz.ch
ETH Zurich

\textsuperscript{3}paul.smith@ait.ac.at
AIT Austrian Institute of Technology GmbH

07.10.2013
ICNP NPSec 2013, Göttingen, Germany
Outline

1. Introduction
   - Objectives
   - SDN and OpenFlow

2. Approach
   - Attack Model
   - STRIDE
   - Attack Trees
   - Combining the Approaches
   - Experimental Setup

3. Results
   - Security Analysis
   - Empirical Testing

4. Recommendations

5. Conclusion
Objectives

- Security analysis of OpenFlow protocol and networks
  - Focus on v1.0.0, but extensible/adaptable methodology
  - Develop model
  - Analyze model
  - Describe attacks

- Empirically demonstrate one or more security issues
  - Develop setup to enable this empirical demonstration

- Suggest potential fixes and mitigations for security issues
Why OpenFlow Security Analysis?

- OpenFlow started as a largely academic endeavour
- But has recently seen increasing deployment in production systems:
  - Google’s OpenFlow WAN
  - Cisco, Juniper, HP products
  - Adoption by cloud hosts and service providers
- But why security?
  - No official security analysis of the protocol itself
  - Research is just catching up (see HotSDN 2013 program)
  - Security is extremely important for production systems, but can be overlooked
Software Defined Networks (SDNs) separate _data plane_ and _control plane_.

OpenFlow implements SDN:
- Switch implements data plane
- _Controller_ implements control plane
- Switch and controller connected with _secure channel over control network_
- Controller installs _flow rules_ on switch
- Flow rule _header fields_ match packet headers
- Packets matching a flow rule have _actions_ performed on them
Three scenarios

- Attacker controls a single client
- Attacker controls multiple clients
- Attacker has access to control network

The first scenario is given greatest consideration

Scenarios where attacker has access to actual secure channel are not considered

- This would involve compromising SSL or TLS, which is outside the scope of this work
STRIDE

- Security modeling methodology
- Types of vulnerabilities modeled by the method[3]:
  - Spoofing
  - Tampering
  - Repudiation
  - Information Disclosure
  - Denial of Service and
  - Elevation of Privilege
- Use data flow diagrams to uncover potential vulnerabilities
  - Models how external data enters into and propagates through system

Figure: Data flow diagram
Attack Trees

- Used to describe and analyze attacks
- Based on *fault tree analysis*[^4]
- Represent prerequisites for attacks
  - Leaf nodes represent actions or events
  - These propagate through AND and OR gates
  - Root node is objective
  - Can calculate various metrics if values for leaf nodes are known

[^4]: [4]
From STRIDE DFDs to Attack Trees

- Data flow diagrams show us potential *vulnerabilities*
  - They show us which components present an *attack surface*
- Attack trees allow these to be developed into practical *attacks*
  - A given objective may have multiple *attack paths*
  - Attack trees help to analyze and optimise attack paths
- These two approaches are complementary
Experimental Setup

- **Mininet** is a virtual network emulation environment
  - Based on Linux *network namespaces*
  - Runs *Open vSwitch* (virtual OpenFlow switch)
- Can emulate performance constraints
  - Bandwidth
  - Latency and jitter
  - This is required to simulate attacks
- Forms the basis of test environment
  - Use *POX* as a controller
Setup Schematics

**Figure:** Network topology for Denial of Service attack demonstration

**Figure:** Network topology for Information Disclosure attack demonstration
Denial of Service I

Figure: Data flow diagram of switch
Denial of Service II

Figure: Close-up of data flow diagram
Denial of Service III

Security Analysis
Empirical Testing

Introduction
Approach
Results
Recommendations
Conclusion

Denial of Service
Against switch
Against controller
Against Flow table
Against OpenFlow Interface and data flow

Denial of service
Generate very high traffic load on interface
Exploit security hole in controller (if present)
Obtain access to multiple client interfaces
Locate security hole in controller software
Develop exploit
Perform processor intensive tasks on several connections
Identify which flow rules are created without wildcards
Identify exact form of flow table entries
Identify hash function used for flow table
Cause hash collisions on flow table

Figure: Denial of Service attack tree with attack path highlighted
Denial of Service IV

Figure: Close-up of highlighted attack path
Information Disclosure I

Figure: Data flow diagram of controller
Information Disclosure II

Figure: Close-up of data flow diagram
Information Disclosure III

Figure: Information Disclosure attack tree with attack path highlighted
Information Disclosure IV

Figure: Close-up of highlighted attack path
Denial of Service

Figure: Number of lost packets vs rule timeout value due to flow table overflow (with control link at 100 Mbps and 1ms latency)
Information Disclosure I

Figure: Distribution of measured times with exact matching flow rules
Figure: Distribution of times with source address and port as wildcards
Denial of Service

- Rate Limiting, Event Filtering, Packet Dropping, Rule Timeout Adjustment
  - Some of them introduced in newer OpenFlow standards
  - Example of usage: large timeouts lighten load on controller but can cause table overflows
- Flow Aggregation
  - Try to reduce load on controller with proactive strategies
- Attack Detection
  - Employ OpenFlow for logically centralized detection
  - Direct flows to specialized monitoring systems
- Access Control - Distributed Firewall
  - ACLs implemented as sets of flow rules on the switch
Information Disclosure

- **Proactive Strategies**
  - Remove response time-state dependency

- **Randomization**
  - Increase variance of measurable response times
  - Clever rule timeout randomization

- **Direct Attack Detection-Mitigation**
  - Exploit bird’s eye view over traffic to detect suspicious patterns
  - Enact counter-measures using direct flow control
Conclusion

- Found potentially problematic issues in OpenFlow, including:
  - Denial of service (i.e. resource depletion)
  - Information disclosure (i.e. timing analysis)

- Newer specifications reflect some of these issues
  - Metering, multiple controllers with fail-over, parallelism
  - But further work is required!

- Demonstrated two different forms of attack
  - Developed test setup (could be used for unit tests)

- Contributions
  - Extensible and adaptable methodology
  - Towards SDN architectures that are more secure by design
Thank you very much for your attention!

Questions?
Other Approaches

- **Attack nets (from Petri nets)**[5]
  - More versatile than DFDs, but also harder to analyse
  - This level of formalism is not needed
  - Less suited to fully asynchronous system
  - Difficult to model system with discrete, fully enumerated states

- **State-based system models**[1, 2]
  - These systems tend to model *control flow* rather than data flow
  - OpenFlow specification does not require any particular control flow
  - Might be useful with a given controller
Denial of Service V

Figure: Denial of service attack tree with attack path highlighted
Denial of Service VI

Figure: Close-up on highlighted attack path
Force another client to reflect traffic or produce response
UDP based traffic
ICMP based traffic
TCP based traffic

ICMP echo request/echo response (ping)
DNS request
NTP
RIP
HTTP request
SSH or telnet

NB: This attack tree should not be considered exhaustive.

Figure: Information disclosure attack tree with attack path highlighted
Information Disclosure VI

Figure: Close-up on highlighted attack path
Figure: Number of lost packets vs timeout value due to flow table overflow (with control link at 10 Mbps and 10ms latency)
Figure: Distribution of times with source address and port as wildcards and asymmetrical delay (delay in control network shorter than in data network)

URL: http://doi.acm.org/10.1145/2000259.2000281

Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack.
Wikipedia.
Fault tree analysis.

Ruoyu Wu, Weiguo Li, and He Huang.
An Attack Modeling Based on Hierarchical Colored Petri Nets.
doi:10.1109/ICCEE.2008.69.