OpenFlow: A Security Analysis

Rowan Klöti[1] Vasileios Kotronis[2] Paul Smith[3]

[1]rkloeti@alumni.ethz.ch
ETH Zurich

[2]vkotroni@tik.ee.ethz.ch
ETH Zurich

[3]paul.smith@ait.ac.at
AIT Austrian Institute of Technology GmbH

07.10.2013
ICNP NPSec 2013, Göttingen, Germany

Outline

Objectives

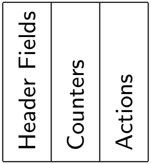
- Security analysis of OpenFlow protocol and networks
  - Focus on v1.0.0, but extensible/adaptable methodology
  - Develop model
  - Analyze model
  - Describe attacks
- Empirically demonstrate one or more security issues
  - Develop setup to enable this empirical demonstration
- Suggest potential fixes and mitigations for security issues

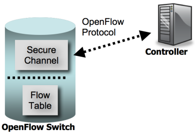
Why OpenFlow Security Analysis?

- 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

SDN and OpenFlow 101

- 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





Attack Model

- 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]:
  - **S***poofing*
  - **T***ampering*
  - **R***epudiation*
  - **I***nformation Disclosure*
  - **D***enial of Service* and
  - **E***levation of Privilege*
- Use *data flow diagrams* to uncover potential vulnerabilities
  - Models how external data enters into and propagates through system

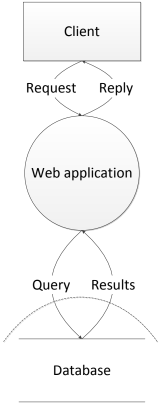


Figure : Data flow diagram

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

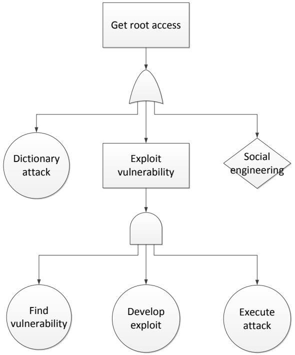


Figure : Attack tree

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

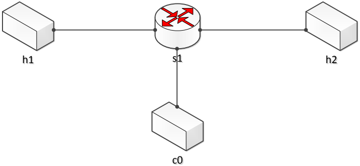


Figure : Network topology for Denial of Service attack demonstration

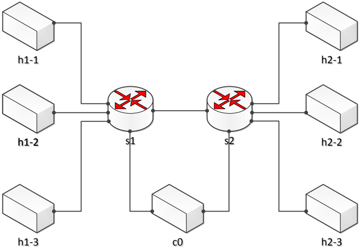


Figure : Network topology for Information Disclosure attack demonstration

Denial of Service I

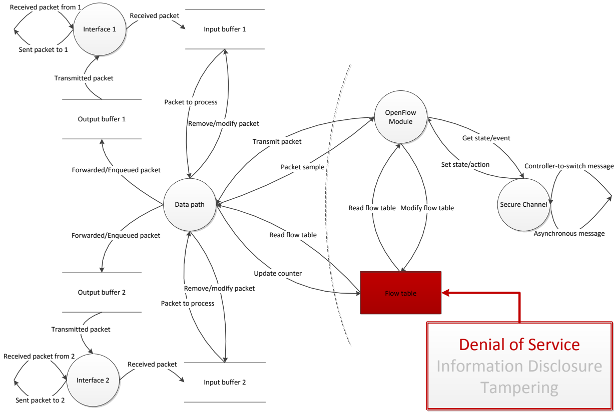


Figure : Data flow diagram of switch

Denial of Service II

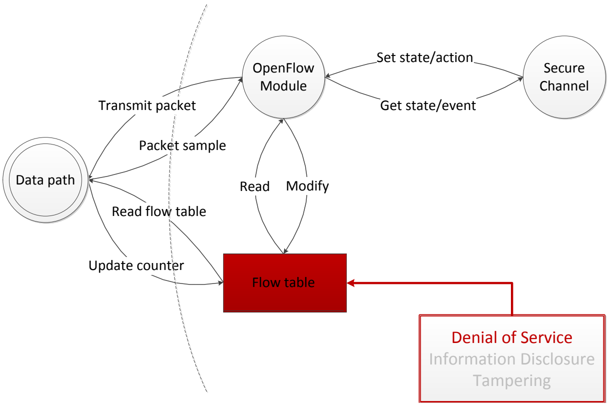


Figure : Close-up of data flow diagram

Denial of Service III

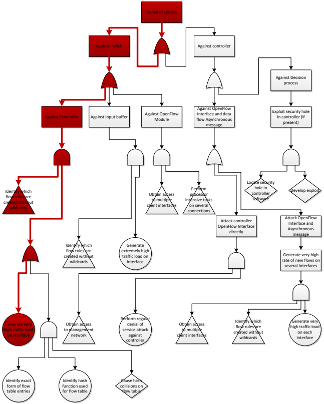


Figure : Denial of Service attack tree with attack path highlighted

Denial of Service IV

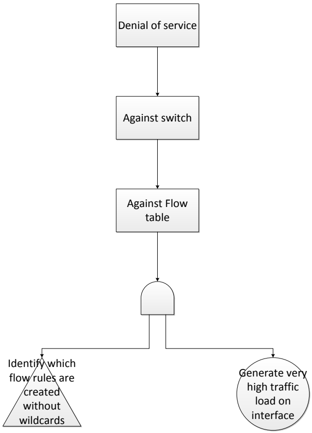


Figure : Close-up of highlighted attack path

Information Disclosure I

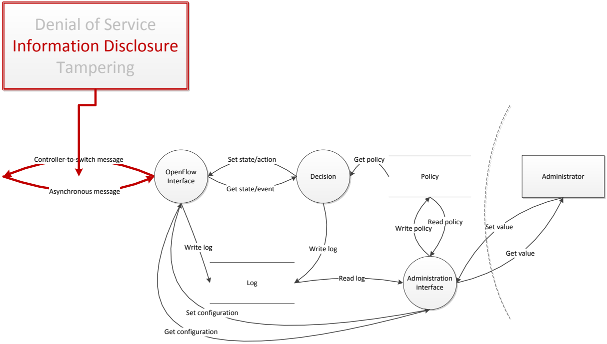


Figure : Data flow diagram of controller

Information Disclosure II

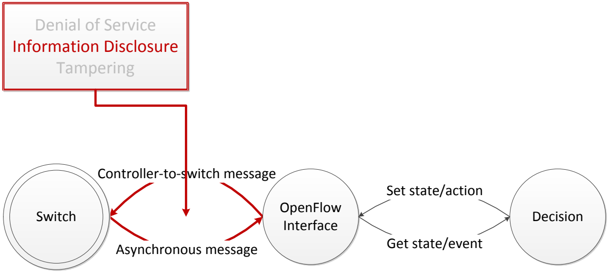


Figure : Close-up of data flow diagram

Information Disclosure III

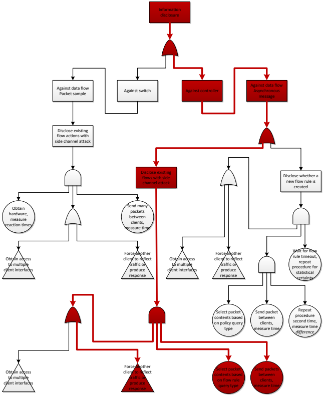


Figure : Information Disclosure attack tree with attack path highlighted

Information Disclosure IV

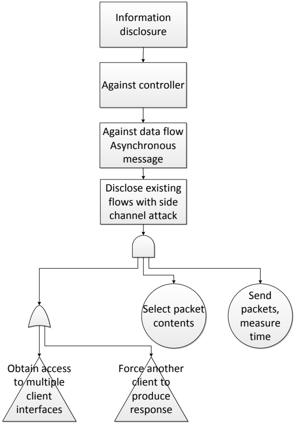


Figure : Close-up of highlighted attack path

Denial of Service

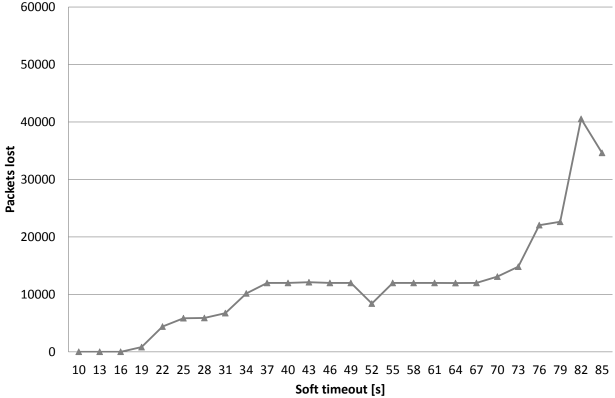


Figure : Number of lost packets vs rule timeout value due to flow table overflow (with control link at 100 Mbps and 1ms latency)

Information Disclosure I

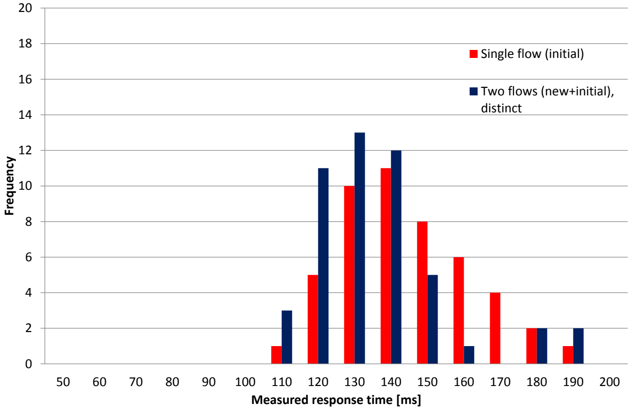


Figure : Distribution of measured times with exact matching flow rules

Information Disclosure II

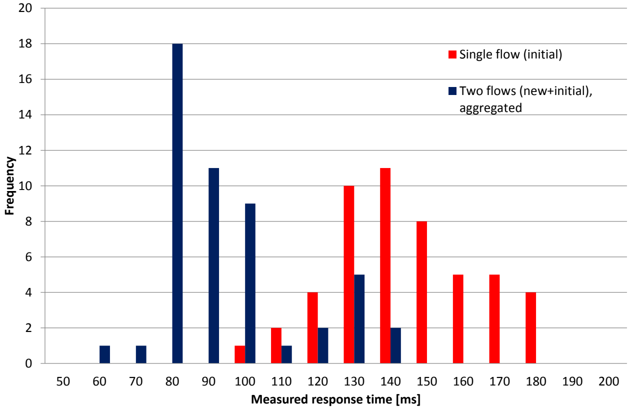


Figure : Distribution of times with source address and port as wildcards

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

Discussion

- 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

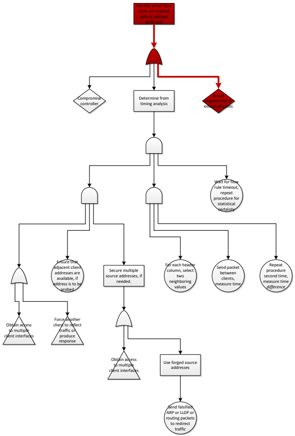


Figure : Denial of service attack tree with attack path highlighted

Denial of Service VI

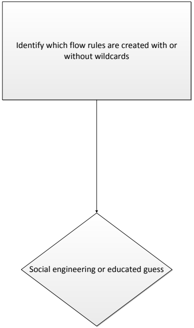


Figure : Close-up on highlighted attack path

Information Disclosure V

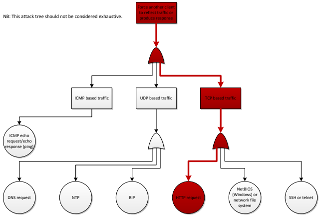


Figure : Information disclosure attack tree with attack path highlighted

Information Disclosure VI

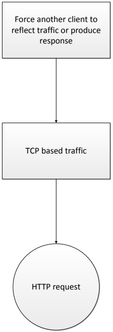


Figure : Close-up on highlighted attack path

Denial of Service (Empirical) II

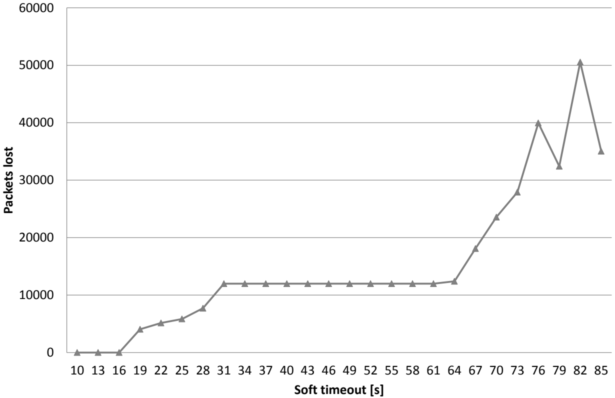


Figure : Number of lost packets vs timeout value due to flow table overflow (with control link at 10 Mbps and 10ms latency)

Information Disclosure (Empirical) III

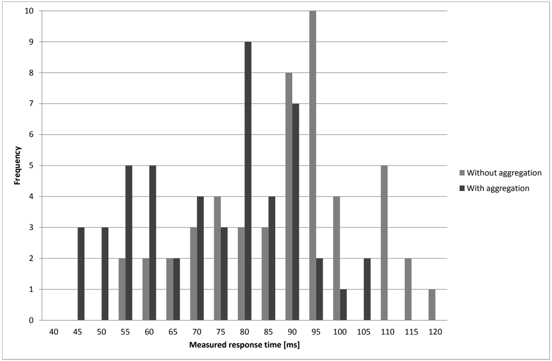


Figure : Distribution of times with source address and port as wildcards and asymmetrical delay (delay in control network shorter than in data network)

References I

O. El Ariss, Jianfei Wu, and Dianxiang Xu.
Towards an Enhanced Design Level Security: Integrating Attack Trees with Statecharts.
In *Secure Software Integration and Reliability Improvement (SSIRI), 2011 Fifth International Conference on*, pages 1–10, june 2011.
doi:10.1109/SSIRI.2011.11.

Omar El Ariss and Dianxiang Xu.
Modeling security attacks with statecharts.
In *Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems*

References II

– *ISARCS*, QoSA-ISARCS '11, pages 123–132, New York, NY, USA, 2011. ACM.
URL: http://doi.acm.org/10.1145/2000259.2000281, doi:10.1145/2000259.2000281.

Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack.
Uncover Security Design Flaws Using The STRIDE Approach, 2006.
URL: http://msdn.microsoft.com/en-gb/magazine/cc163519.aspx.

References III

Wikipedia.
Fault tree analysis.
http://en.wikipedia.org/wiki/Fault_tree_analysis.
Accessed on 02.04.2013.

Ruoyu Wu, Weiguo Li, and He Huang.
An Attack Modeling Based on Hierarchical Colored Petri Nets.
In *Computer and Electrical Engineering, 2008. ICCEE 2008. International Conference on*, pages 918–921, dec. 2008.
doi:10.1109/ICCEE.2008.69.