Title goes in report.tex

Bonnie Eisenman, Michael Kranch and Anna Kornfeld Simpson

COS 597E Software Defined Networks
14 January 2014

Abstract

This paper looks at the security concerns and potential vulnerabilities in Software Defined Networking (SDN). In particular, we examine the attack vector of remotely introducing a switch into the networking path, something not possible without physical access to the switching site in a traditional network. We also look at the increased risk to the entire network from a single, compromised switch due to the centralized nature of SDN . First, we provide an overview of SDN, the current status of security in SDN controllers, and discuss the threat models for our attack. We then introduce our roque switch and present several attacks on various controller. We also discuss additional attacks possible by a rogue switch and examine additional vulnerabilities to a SDN from a single compromised switch. Finally, we conclude with some recommendations for hardening SDN controllers and limiting the area of attack.

1 Introduction

Paragraph about SDN - the various meanings of SDN and that we are discussing the traditional controller switch networking.

Not sure of difference between intro / background anymore as Anna mentioned Friday.

Things to discuss – Controllers —-Various Controller. That we are doing testing on Ryu and Pox (possibly OpenDaylight but its more complicated) —-Switches - In this context, a switch means either an OpenFlow capable hardware switch or a completely software defined switch (OpenVSwitch) —-Need to Discuss Proactive verses Reactive Flow Installation - While the majority of the attacks presented in this paper would be more effective in a reactive Flow Installation configuration, they would still be

Threat Model - Need to make sure we specifically discuss current security in switch - using TLS certs, who supports, etc. In section 2, we will.... In Section 3, etc. In Section 4, etc. Finally

2 Background

2.1 Software Defined Networking

2.2 Security in SDN

3 Creating a Software Defined Switch

In this section, we present our roque switch and

3.1 OpenFlow Protocol

OpenFlow is the protocol used to communicate between the controller and its switches. An OpenFlow packet header is simply an 8 byte packet with the first byte used to communicate version, the second byte for type, third and fourth byte for message length followed by a four byte Transaction ID (see Figure 1). The type is of a subset of 19 possible types, most of which are of type request (e.g. 16 = Stats Request) with the subsequent reply (e.g. 17 = Stats Reply). There are couple key point to note when dealing with the OpenFlow protocol. First, the transaction ID is used to correlate incoming OpenFlow messages with their appropriate responses much in the same way TCP uses SYN and ACK flags. For example, a features reply will (generally) not be accepted by the controller unless it contains the corresponding transaction ID from the features request. This protocol is also a two-way communication scheme and not simply switch replies to controller request. Several message types, to include packet input events, are switch initiated communications to the controller.

3.2 Controller Connection Sequence

While the specific initiation sequence varies by controller , there are several required command to initiate a switch to controller connection[1]. The switch to controller connection simply starts by opening a TCP connection to the controller on the configured port (default 6634)[2]. After opening the connection, the switch sends a Openflow Hello message (0) to the controller. The controller then responds with a Hello message and a Features Request (5) message to which the switch responds with a Features Reply (6). The controller then sends a Set Config message (9) that does not trigger a reply. Ryu follows this message with a Barrier Request (18), and OpenDaylight follows it with a Get Config Request (7) to ensure switch configs are set appropriately. OpenDaylight also sends a Flow Modification message (14) to delete any previously installed flows on the switch.

The above commands are all that is needed for the initial controller to switch connection. The switch then proceeds to send several link layer neighbor discovery protocol messages as Packet Input Notification (10) messages to the controller including Neighbor Solicitation and Router Solicitation messages. The switch also sends s Multicast Listener Report (part of the Multicast Listened Discovery Protocol Version 2) to the controller. Finally, the switch begins periodically (approximately every 3 seconds) sending an OpenFlow Echo Request (2) to the controller as a form of a keep alive with the controller.

3.3 Our Rogue Switch Utility

Our Roque Switch Utility mimics the initial connection sequence to a controller in order to facilitate testing of switch and controller vulnerabilities. Our roque switch utility does not handle the actual routing of traffic and is instead simply used to disrupt controller and switch traffic flow[3]. The bulk of our utility is a OpenFlow message parser that handles and appropriately responds to controller messages in the correct message format. These messages are generally hardcoded mimicked responses from previously captured live switch communication with minimal dynamic pieces (the transaction id is dynamically assigned for instance).

4 Disrupting an SDN with a Fake Switch

There's a file called attacks.tex for describing our various attacks; we could combine the results of the attacks we implemented here, or put them separately.

4.1 Executed Attacks

We conducted several attacks on various controller utilizing our rogue switch utility. One key point here is that the method of implementation of the controller (proactive or reactive) as well as the applications running on the

[1] We specifically tested our roque switch on the Pox, Ryu and OpenDaylight controllers. Based on our testing and the OpenFlow specifications, we believe all controllers require this small subset of initialization commands but differences might exist based on implementation

[2] As previously discussed, there is no authentication for this connection supported by any tested controller outside of certificate authentication only utilized with OpenVSwitch — we need to verify that this statement is true. Only need to test our fakeswitch works outside of same box i.e. not only 127.0.0.1

[3] A more advanced utility including routing is discussed in our additional attack section as well as our future work

controller will significantly effect the results of these attacks. We utilized the basic layer 2 mac address learning example from each controller in our setup [4]. A more complicated controller (e.g. one that does more than simply add the mac address as a flow) would see its performance decrease more significantly than in our example test cases.

4.1.1 DOS on the Controller

The first attack we attempted was to see if a single switch could overall the controller by sending too By sending requests to the controller, the switch could attempt to overload it. One could fine-tune the control messages to maximize churn or CPU consumption in the controller.

However, our results show that a TCP congestion control successfully prevents a single switch from denying service to the controller; instead the extra packets from the switch are dropped.

What happens if in a network with n actual switches, the controller receives hello messages from n^2 switches, for example? This works around the throttling problem where a controller may simple rate-limit each switch.

4.2 Dropping traffic

A compromised switch could simply drop packets sent to it, thus creating a service interruption. However, a controller would probably notice this quickly – it would appear as though the switch had failed, and automated recovery mechanisms would be initiated.

4.3 Cloning or diverting traffic

A compromised switch could ensure that traffic would be routed through an adversarys middlebox, or could clone traffic and send the cloned stream to the adversary. This would be more difficult for the controller to detect, unless it caused considerable slowdown. This attack could conceivably be used for purposes such as the NSAs MUSCULAR program, which intercepts traffic as it flows between private data centers [5].

4.4 DoS other switches

By injecting traffic, a switch could deliberately attempt to overload neighboring switches TCAMs or otherwise DoS them.

4.5 Advertising false host attachments

A compromised switch can send the controller packet-in events with false packets, which falsely claim to have a particular MAC address attached to them. This could lead the controller to believe that the compromised switch should receive traffic intended for the host. While this would soon lead to packet loss and a noticeable disruption, it would nevertheless allow a switch to eavesdrop when it would not normally be in a position to do so.

4.6 Falsify measurement reports

A switch may return false results in response to a read-state measurement message, thus causing the controller to behave irrationally. For example, a switch could falsify or hide a DoS attack, elephant flows, etc.

4.7 Ignoring rules

A switch could simply ignore flow-table modification requests. For example, dropping packets will be noticed quickly; but a switch could allow packets to pass through that should have been dropped. Because of the difficulty of querying flow table state, the controller may not become aware of this. A compromised switch could thus operate in stealth mode and the inconsistent flow table might only be noticed once the switch allows a DoS attack to pass through, for example.

4.8 Modifying VLAN tags

4.9 Reporting flow mods to an adversary

An adversary could hope to learn about network traffic by observing flow mods. Certain flow mods might indicate that the controller

[4] We used l2_learning as our Pox controller and l2_switch as our Ryu controller. We modified the hard flow timeout on both to 1 to better test controller delay.

has, for example, detected an intrusion, or that a specific host has connected to the network in a certain location. Thus, just knowing what flow mods are being issued by the controller could be a source of interesting information for an adversary.

4.10 Even more...

5 Related Work

The related work might be long, so let's put it in a separate file, related.tex This is related.tex - I've already added all the links from the email thread to the bibliography, we can remove the ones we won't plan to use. TODO for Anna... look up the two security papers we read in class, add them to the bibliography and talk about them here.

6 Conclusion

The conclusion might be short enough to be in report.tex - we can add other sections before it as necessary, and the bibliography (in bibliography.tex) will follow.

References

[1] Benton et. al. "OpenFlow Vulnerability Assessment" *ACM SIGCOMM Hot Topics in Software Defined Networking (HotSDN)*, 2013.

[2] Cisco. "Multiple Vulnerabilities in Cisco TelePresence Multipoint Switch" 11 July 2012 `tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20120711-ctms`

[3] Constantin, Lucian. "Cisco patches vulnerabilities in some security applications, switches, and routers." *InfoWorld* 10 October 2013, `http://www.infoworld.com/d/security/cisco-patches-vulnerabilities-in-some-security-appliances-switches-and-routers-228551`

[4] dugsong et. al. "dpkt" `code.google.com/p/dpkt`

[5] Gellman et. al. "How we know the NSA had access to internal Google and Yahoo cloud data" *Washington Post, The Switch*, 4 November 2013, `http://www.washingtonpost.com/blogs/the-switch/wp/2013/11/04/how-we-know-the-nsa-had-access-to-internal-google-and-yahoo-cloud-data/`

[6] Hecker, Artur. "Carrier SDN: Security and Resilience Requirements" 15 November 2013 `http://www.ikr.uni-stuttgart.de/Content/itg/fg524/Meetings/2013-11-15-Muenchen/12_ITG524_Muenchen_Hecker.pdf`

[7] "OpenFlow Packet Protocol." *OpenFlow*, `archive.openflow.org/wk/images/c/c5/Openflow_packet_format.pdf`

[8] Rowe, Mark. "Python Packet Capture and Injection Library" `pycap.sourceforge.net`

[9] Shin and Gu. "Attacking Software-Defined Networks: A First Feasibility Study" *ACM SIGCOMM HotSDN*, 2013.

[10] Stretch, Jeremy. "Understanding TCP Sequence and Acknowledgement Numbers" 7 June 2010, `http://packetlife.net/blog/2010/jun/7/understanding-tcp-sequence-acknowledgment-numbers/`