Implementation of an SSL Proxy in a Software Defined Network Aboard a Naval Ship to Monitor Internet Traffic

Abstract

Cyber Security is essential aboard US Naval ships. With a variety of people taking part in the ship's network, browsing the web or otherwise, measures must be taken to ensure oversight of each user. The problem is that with the conventional network used today, a network administrator or information technology professional cannot know how many devices are aboard the ship and connected, much less their activity on the network. The solution I used was the implementation of a SSL Proxy in a new network paradigm known as Software Defined Networking (SDN). An SDN simplifies routers, switches and other network devices by adding a centralized controller which allows full visibility of every device on the network, dynamic programmability of traffic flows and full control over traffic content [7]. With this new ability of control over the network, I wrote an application to access each device's web traffic. Using a "man-in-the-middle" or more accurately "controller-in-the-middle" technique to create my proxy within this network paradigm, I was able to use the SDN's capabilities to have full knowledge of every device's existence and web traffic. Thus, a network administrator can now use my software to monitor and maintain a secure network aboard the ship.

Keywords-- ***Software-defined Network, SSL proxy, Man-in-the-middle, cyber security***

1. Introduction

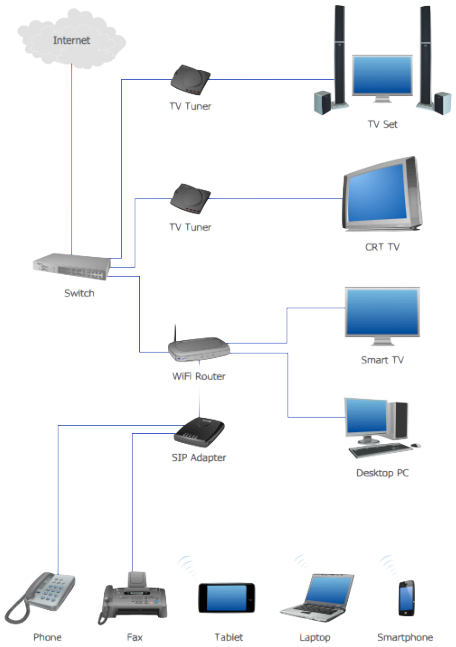
A naval warship is a complex system of people with a variety of skills and backgrounds that accomplish all of the tasks required to achieve their strategic objectives. This spectrum of people, ranging from admirals to maintenance crews, creates a unique problem of satisfying everyone's needs without sacrificing the security of the ship as a whole. One basic need is a connection to the outside world using the internet. Web surfing can range from a crewmember keeping in touch with his family via email or an admiral accessing top secret intelligence needed to make mission critical decisions. Therefore, it is essential for information technology administrators aboard naval ships to enable access to the internet, and more importantly, monitor the network traffic to ensure the integrity of the network. The specific problem is that it is difficult to secure onboard networks without eliminating their connectivity inside and outside the ship [10]. Simply put, if they allow open access to the internet using old hardware defined systems, they can not see what information people are sending or receiving over the internet while onboard the ship. The ability to identify and monitor each connection to networks off the ship is an unsolved problem [10]. The solution I devised was to implement an SSL proxy in a relatively new network paradigm known as a Software Defined Network (SDN) that enables this very concept to be executed. This is essential to providing a secure network for many reasons, one of which is malware detection. Malware creates a specific kind of web traffic that allows its presence to be known but the contents are often

encrypted. Using my combination of an SDN and an SSL proxy allows more security in situations such as malware detection.

Software Defined Networking (SDN) is still a very young concept; Google and Facebook have developed SDNs to be used in their data centers [11]. They are also being used as a research tools in universities and postgraduate schools across the country [10]. The rising popularity of SDNs stems from their versatility. They allow unprecedented control and visibility into a network without sacrificing significant efficiency or speed. With the use of the software defined network I created, an administrator can have full access to the contents of the traffic, the paths in which the rates of the data flow and the information about each device that is connected. Furthermore, to be able to monitor the contents of traffic, I set up an SSL proxy, which is an approach to capture traffic on an SSL connection. I set up the controller to act as a “man-in-the-middle”-- spoofing the destination web server to the client and then spoofing the client (only changing the source hardware and software addresses to receive the response) to the web server. This way I intercepted the traffic coming in and out of the network. Then I used a technique first developed by a cyber security expert by the pseudonym of Moxie Marlinspike to decrypt the SSL connection and capture the traffic [10]. Through my SSL Proxy technique, an administrator can intercept all traffic coming to and from every device on the network and analyze its integrity as a trusted or malicious user on the ship. My solution is practical because all the necessary components are normal networking equipment.

2. Creating an SDN and SSL proxy

Aside from the users' devices, a SDN is made up of four additional devices: switches, routers, modems and controllers. Except for a controller, these devices are already in place in a conventional network design. To set up and monitor the ship's internet users, I ran a controller as a web server with Ryu (v3.12), a component-based SDN framework [1]. Ryu uses Openflow (v1.1) as the communication protocol to the forwarding plane of a network switch or router. By decoupling the network control and forwarding functions, network control was directly programmable and the underlying infrastructure was abstracted for applications and network services [8]. The controller was able to route traffic in the network by installing flow modifications on the switches themselves. Normally, traffic flows from the hosts to the switches and the switches to the router before finally leaving to the outside world through a modem. (See fig 1)



Topology of Equipment on a Standard Network and the Data Flow (Fig 1.) [12]

However, for the purpose of this project, the only traffic I routed to the controller were packets utilizing the Hyper Text Transfer Protocol (HTTP) or Hyper Text Transfer Protocol Secure (HTTPS) on the Secure Sockets Layer (SSL). These protocols are used by devices connecting to the internet to interact with web servers. The only difference between the two is SSL uses an encrypted connection, whereas HTTP transmits data as plain text. Using Openflow API calls, I was able to route HTTP and HTTPS traffic from the SDN switches to the controller, which enabled the controller to impersonate the client, using different source hardware and software addresses to receive the responses from the web server. This way, the responses coming from the web server would be sent back to the controller rather than to the client. The Openflow libraries allowed me to craft packets from the controller with the information given by the client's traffic. Once the controller received the responses from the web server, the controller forwarded them to the client, but this time I spoofed the software and hardware addresses of the server. All that the server received was traffic that appeared to be from the client and client received traffic that appeared to be from the web server. The client and the server could not detect the controller acting as a proxy. The controller inspected the contents of each packet before it forwarded them to the correct destinations. The controller was now a "man-in-the-middle" and could see all traffic going into the network and coming out before it reached its final destination.

Inserting a proxy into an SSL connection was slightly more complicated. I used a technique developed by a cyber security expert under the pseudonym of

Moxie Marlinspike. SSL is never used without making a regular HTTP connection first [5] .The client is redirected to use an HTTPS page via an HTTP 302 response code. I intercepted the transition from the unsecured connection (HTTP) to a secure one (SSL or HTTPS). That way I could use the same technique I used for HTTP connections without having to authenticate my controller's information to the client or the server. The exchange is more easily visualized in figure 2 below.

Figure (2). [5] SSL Hijack in an SDN Environment

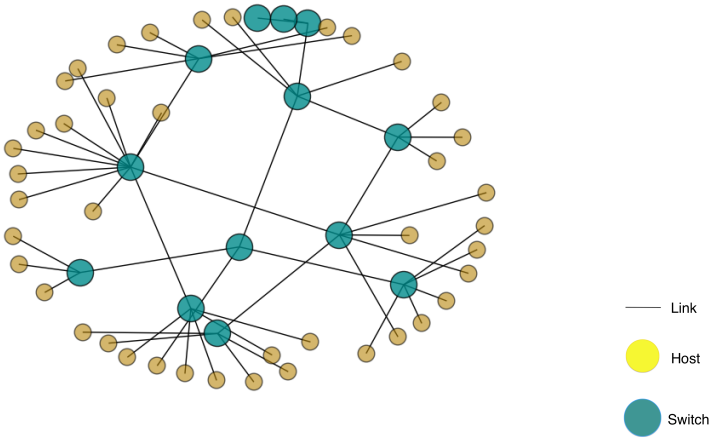
Now the controller had access to the contents of every HTTP and HTTPS connection on the network. I could inspect, change and analyze every packet to determine the user's integrity.

3. Testing Code on Virtual Network

To test this theory, I used Mininet (v2.2) [9], an open-source network simulator, to build a virtual network. With this software, I created a test network with different topology designs to test my technique. I logged onto hosts on the network and sent HTTP and HTTPS requests to the outside world and observed how the controller and my SSL proxy handled the traffic. With this technique, I saw the results of my SSL proxy were successful. It clearly showed the ability to identify

and monitor each user on the network. I could see information about the device, where they were connected, and the contents of their traffic.

To demonstrate my ability to have full knowledge of every device on the network and the switch it's connected to, I wrote a topology detection program that maps all devices on a network as seen in figure 3 below. Blue dots indicate switches and yellow dots indicate hosts or clients on the network.



Topology of devices on virtual network (Fig. 3)

And finally, to get a proof-of-concept of my SSL proxy, I dumped the contents of all incoming and outgoing traffic into a text file. Here is a sample output:

(Packet information)

Date: Fri, 19 Jun 2015 02:15:00 EST

Content-Type: text/html

Content-Length: 70619

Last-Modified: Thu, 18 Jun 2015 05:30:30 EST

Connection: keep-alive

ETag: "55825776-113db"

Accept-Ranges: bytes

(Here we see the HTML code start)

<!DOCTYPE html>

<html lang="en" id="facebook" class="no_js">

<head><meta charset="utf-8" /><script>function envFlush(a){function b(c){for(var d in

a)c[d]=a[d];}if(window.requireLazy){window.requireLazy(['Env'],b);}else{window.Env=window.Env||{};b(window.Env);}}envFlush({"ajaxpipe_token":"AXiJb4dl-SoYbaL-","lhsh":"UAQGvb2rd","khsh":"0`sj`e`rm`s-0fdu^gshdoer-0gc^eurf-3gc^eurf;1;enbtldou;fduDmdldourCxO`ld-2YLMIuuqSdptdru;qsnunuxqd;rdoe-0unjdojnx"});</script>

…

(Full output was not included to comply with the page limit)

My software includes over 2,000 lines of code. Here is a small sample of the code I wrote to accomplish my project:

```
##################################
#  ARP and ICMP Packet Handlers  #
##################################
  def _handle_arp_rq(self, dst_ip):
      pkt = packet.Packet()
      pkt.add_protocol(ethernet.ethernet(ethertype=0x806,\
                                         dst='ff:ff:ff:ff:ff:ff',\
                                         src=self.hw_addr))
      pkt.add_protocol(arp.arp(opcode=arp.ARP_REQUEST,\
                               src_mac=self.hw_addr,\
                               src_ip=self.ip_addr,\
```

```
                        dst_mac='00:00:00:00:00:00',\
                        dst_ip=dst_ip))
    self._flood_packet(pkt)


def _handle_icmp_reply(self, pkt_icmp, pkt_ipv4, datapath):
    if pkt_icmp.type != icmp.ICMP_ECHO_REPLY: return
    if pkt_ipv4.dst != self.ip_addr:
        print(pkt_ipv4.dst, self.ip_addr)
        return
    print("------------------------")
    print("PING RECEIVED THANK GOD")
    print(pkt_ipv4.src)
    print(pkt_ipv4.dst)
    print(datapath.id)
    print("------------------------")
#Finish ARP redesignation
def _handle_arp_reply(self, pkt_arp, port, dpid):
    if pkt_arp.opcode == arp.ARP_REPLY and pkt_arp.dst_mac == self.hw_addr:
        if pkt_arp.src_ip not in self.active_ips:
            print("ARP from " + pkt_arp.src_ip + "\n")
            self.active_ips[pkt_arp.src_ip] =  [dpid, port]
            self.arp_table[pkt_arp.src_ip] = pkt_arp.src_mac
            if self.topology:
                self.draw_graph(1, draw=True)
    if pkt_arp.opcode == arp.ARP_REQUEST and pkt_arp.src_ip != self.ip_addr:
            if pkt_arp.dst_ip not in self.arp_table: return
        #construct and send ARP reply
        reply_pkt = packet.Packet()
        reply_pkt.add_protocol(ethernet.ethernet(ethertype=0x806,\
                                    dst=pkt_arp.src_mac,\
                                    src=self.arp_table[pkt_arp.dst_ip]))
        reply_pkt.add_protocol(arp.arp(opcode=arp.ARP_REPLY,\
                        src_mac=self.arp_table[pkt_arp.dst_ip],\
                        src_ip=pkt_arp.dst_ip,\
                        dst_mac=pkt_arp.src_mac,\
```

```
                        dst_ip=pkt_arp.src_ip))
            print("Responded to ARP Request: " )
            print("Gave [" + pkt_arp.src_mac + "," + pkt_arp.src_ip + "]" +\
            "[" + pkt_arp.dst_ip + "," + self.arp_table[pkt_arp.dst_ip] + "]")
            self._send_packet(reply_pkt, self.dpids[int(dpid, 16)])
##############################
#  How to send out packets  #
##############################

  def _flood_packet(self, pkt):
      for dpid in self.dpids:
        datapath = self.dpids[dpid]
        ofproto = datapath.ofproto
        actions = [datapath.ofproto_parser.OFPActionOutput(ofproto.OFPP_FLOOD)]
          pkt.serialize()
        out = datapath.ofproto_parser.OFPPacketOut(datapath=datapath, buffer_id=0xffffffff,\
                    in_port=ofproto.OFPP_CONTROLLER, actions=actions,\
                    data=pkt.data)
        datapath.send_msg(out)


  def _send_packet(self, pkt, datapath=None):
   #Pick a random datapath to send from
     if not datapath:
     datapath = self.dpids[self.dpids.keys()[randint(0,len(self.dpids)-1)]]
     ofproto = datapath.ofproto
   pkt.serialize()
   ether_pkt = pkt.get_protocol(ethernet.ethernet)
   if ether_pkt.dst in self.mac_to_port[datapath.id]:
     print("Sending packet out: " + `self.mac_to_port[datapath.id][ether_pkt.dst]`)
        actions = [datapath.ofproto_parser.OFPActionOutput(self.mac_to_port[datapath.id]\
                    [ether_pkt.dst])]
        out = datapath.ofproto_parser.OFPPacketOut(datapath=datapath, buffer_id=0xffffffff,\
                                    in_port=ofproto.OFPP_CONTROLLER, actions=actions,\
                                    data=pkt.data)
        datapath.send_msg(out)
##########################
```

```
#  How to create a flow  #
##########################

  def _create_icmp_flow(self,datapath):
      ofproto = datapath.ofproto
      ofproto_parser = datapath.ofproto_parser
      nw_dst = struct.unpack('!I', ipv4_to_bin(self.ip_addr))[0]
      match = datapath.ofproto_parser.OFPMatch(dl_type=0x800, nw_dst=nw_dst)
      actions = [datapath.ofproto_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER)]
      mod = datapath.ofproto_parser.OFPFlowMod(
            datapath=datapath, match=match, cookie=0, command=ofproto.OFPFC_ADD,
            idle_timeout=0, hard_timeout=0, actions=actions,
            priority=0xFFFF)
      datapath.send_msg(mod)

  def _create_lldp_flow(self, datapath):
      ofproto = datapath.ofproto
      ofproto_parser = datapath.ofproto_parser
      if ofproto.OFP_VERSION == ofproto_v1_0.OFP_VERSION:
         #Add LLDP Rule
         match = datapath.ofproto_parser.OFPMatch(dl_type=0x88cc)
         actions = [datapath.ofproto_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER)]
         mod = datapath.ofproto_parser.OFPFlowMod(
                  datapath=datapath, match=match, cookie=0, command=ofproto.OFPFC_ADD,
                  idle_timeout=0, hard_timeout=0, actions=actions,
                  priority=0xFFFF)
         datapath.send_msg(mod)

  def _create_arp_flow(self, datapath):
      ofproto = datapath.ofproto
      ofproto_parser = datapath.ofproto_parser
      match = datapath.ofproto_parser.OFPMatch(dl_type=0x0806)
      actions = [datapath.ofproto_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER)]
      mod = datapath.ofproto_parser.OFPFlowMod(
            datapath=datapath, match=match, cookie=0, command=ofproto.OFPFC_ADD,
            idle_timeout=0, hard_timeout=0, actions=actions,
```

```
            priority=0xFFFF)
        datapath.send_msg(mod)

#######################################
#  LLDP Portion - Topology Detection  #
#######################################
    def close(self):
        self.is_active = False
        if self.link_discovery:
            self.lldp_event.set()
        self.link_event.set()
        hub.joinall(self.threads)


    def _register(self, dp):
        """
        Takes the datapath and registers
        it as a switch.  This is how
        the controller represents the
        switches.
        """
        assert dp.id is not None

        self.dpids[dp.id] = dp
        if dp.id not in self.port_state:
            self.port_state[dp.id] = PortState()
            for port in dp.ports.values():
                self.port_state[dp.id].add(port.port_no, port)

    def _unregister(self, dp):
        """
        This function is called when a switch
        dies. It helps with the book-keeping
        and clean up.
        """
        if dp.id in self.dpids:
            del self.dpids[dp.id]
```

```
        del self.port_state[dp.id]

    def _get_switch(self, dpid):
        """
        Returns the switch representation
        of the datapath id.
        """
        if dpid in self.dpids:
            switch = Switch(self.dpids[dpid])
            for ofpport in self.port_state[dpid].values():
                switch.add_port(ofpport)
            return switch

    def _get_port(self, dpid, port_no):
        """
        Returns the controller's representation of a port.
        """
        switch = self._get_switch(dpid)
        if switch:
            for p in switch.ports:
                if p.port_no == port_no:
                    return p

    def _port_added(self, port):
        """
        Adds the port to a list of ports used to
        connect two switches.
        """
        lldp_data = LLDPPacket.lldp_packet(
            port.dpid, port.port_no, port.hw_addr, self.DEFAULT_TTL)
        self.ports.add_port(port, lldp_data)
        # LOG.debug('_port_added dpid=%s, port_no=%s, live=%s',
        #           port.dpid, port.port_no, port.is_live())

    def _link_down(self, port):
        """
        Creates an event that will tell the controller
```

```
        that the link state has changed so that the
        controller can reflect that in its representation.
        """
        try:
            dst, rev_link_dst = self.links.port_deleted(port)
        except KeyError:
            # LOG.debug('key error. src=%s, dst=%s',
            #           port, self.links.get_peer(port))
            return
        link = Link(port, dst)
        self.send_event_to_observers(event.EventLinkDelete(link))
        if rev_link_dst:
            rev_link = Link(dst, rev_link_dst)
            self.send_event_to_observers(event.EventLinkDelete(rev_link))
        self.ports.move_front(dst)

    def _lldp_handler(self, msg):
        """
        Will react to LLDP packets by parsing them and creating
        events so that the controller knows to update its topology
        representation.
        """
        try:
            # Attempt to parse the packet as an LLDP packet
            # Will fail if it is not an LLDP packet
            src_dpid, src_port_no = LLDPPacket.lldp_parse(msg.data)
        except LLDPPacket.LLDPUnknownFormat as e:
            # This handler can receive all the packets which can be
            # not-LLDP packet. Ignore it silently
            #print("Not LLDP Packet")
            return

...
```

4. Conclusion

Software defined networks (SDNs) are just beginning to be widely used. Along with more widespread use comes more opportunities for security breaches and thus the need to create "secure" software defined networks (SSDNs). Security flaws need to be identified, networks need to be monitored and patches need to be discovered. I have written code to locate and monitor every user in a network on a Navy ship, but any IT professional for any group of users can use my program. The next step is automating the analysis of traffic, so that a program can flag certain malicious users on the network. This automation will need to be specific for each network because each network's traffic is unique. So either the program has to be written for a specific type of network, such as a naval warship, or the program will have an aspect of machine learning to be able to adapt to any given environment. For my next project, I plan to write the software that can adapt to any given network so that there is a universal model. My program will analyze data flow on a network and understand the difference between normal traffic and anomalous traffic. Any administrator could then see which users may be conducting suspicious behavior and at the very least provide further analysis to determine if they are doing anything wrong.

5. References

[1] *Ryu SDN Framework*. Ryu SDN Framework Community, 2015.

[2] Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey."*proceedings of the IEEE* 103.1 (2015): 14-76.

[3] Jarraya, Yosr, Taous Madi, and Mourad Debbabi. "A survey and a layered taxonomy of software-defined networking." (2014): 1-1.

[4] McKeown, Nick, et al. "OpenFlow: enabling innovation in campus networks."*ACM SIGCOMM Computer Communication Review* 38.2 (2008): 69-74

[5] Sanders, “Understanding Man-In-The-Middle Attacks - Part 4: SSL Hijacking,” *WindowSecurity.com*, Sep-2010. [Online]. Available at: http://www.windowsecurity.com/articles-tutorials/authentication_and_encryption/understanding-man-in-the-middle-attacks-arp-part4.html. [Accessed: 2015].

[6] J. Stewart, “Limited online access stresses sailors at sea,” *Navy Times*, 2012. .

[7] Parker, T.; Johnson, J.; Tummala, M.; McEachen, J.; Scrofani, J., "Identifying congestion in software-defined networks using spectral graph theory," in *Signals, Systems and Computers, 2014 48th Asilomar Conference on* , vol., no., pp.2010-2014, 2-5 Nov. 2014

[8] *Openflow*. Open Networking Foundation, 2015.

[9] *Mininet*. Mininet Team, 2015.

[10] CDR Staples, Z. Director, Navy Center for Cyber Warfare at the Naval Postgraduate School, Monterey, CA. (2015, July 20). Personal interview.

[11] A. Vahdat, 'A look inside Google’s Data Center Networks', *Google Cloud Platform Blog*, 2015. [Online]. Available: http://googlecloudplatform.blogspot.com/2015/06/A-Look-Inside-Googles-Data-Center-Networks.html. [Accessed: 12- Sep- 2015].

[12] Concept Draw, *Network Topology Graphical Examples*. 2015.