QUARTERLY TECHNICAL PROGRESS REPORT

Shakedown Experimentations and Prototype Services on Scalable, Agile, Robust, and Secure Multi-Domain Software Defined Networks

Report Period: Jan. 1, 2015 – Mar. 31, 2015

Technical Point of Contact
Professor Matt Bishop
University of California
Dept. of Computer Science
Watershed Sciences, Rm 2209

Professor S. J. Ben Yoo
University of California
Dept. of Electrical and Computer Engineering
Kemper Hall, Rm 3179
Davis, California 95616
Tel: 530-752-7063
Fax: 530-752-8428
E-mail: bishop@ucdavis.edu, sbyoo@ucdavis.edu

Table of Contents

I. Major Accomplishments

A. Milestones Achieved

Table 1 summarizes the status of completion for the different milestones indicated in Year 2 period. This report discusses in particular the technical progress related to tasks and milestones for the period January 1, 2015 – March 31, 2015.

Table 1. List of milestones achieved with status of completion.

Task	Milestones	Status
1	Given an OpenFlow controller server, how can an attacker gain access to that server, and how can she reconfigure it to give her desired control over the OpenFlow controller	COMPLETED

The following sections will describe in details the studies and findings related to the tasks mentioned above. In particular, during the recent three months of the project, our research team focused on security problems in SDN network infrastructures. The following sections will describe in details the studies and findings related to each of the activities above.

B. Deliverables Made

The deliverables include:

1. Three poster files have been presented at GEC'22, showing the project progress and our demos.
2. Two live demos of security threats to an SDN network infrastructure. The first demonstrated diverting traffic to a third monitoring host that could read, alter, inject, and/or delete packets moving through an SDN switch. This demo used data collected from a live network and replayed at GEC22. The second demonstrated a denial of service attack originating from the SDN router based on commands from the controller. The demo used data collected from a mininet simulation.
3. Two published papers.

II. Description of Work Performed During Last Quarter

A. Activities and findings

Security policies have three types of constraints: those based on a need to keep traffic confidential, those based on a need to keep traffic unaltered without authorization, and those based on a need for packets sent over the network to arrive at the intended destination. We therefore focused on whether these properties could be disrupted on a software-defined network more easily than on a regular network. A critical component of such networks is the controller, which manages the SDN switches and can reprogram routes. The controller is composed of special software to manage the network, and is run on a commodity system, for example a Linux system. We asked what a compromised controller could do to violate the properties defined above.

We first set up several configurations of a software-defined network and attempted to divert or block traffic. After some preliminary analysis, we focused on two specific examples.

In all these examples, we used an isolated network to avoid interference with production traffic. We then created traffic between a user system (a laptop) and a server system. Each system had *wireshark*, a network sniffing program, monitoring all incoming and outgoing packets. All packets captured were saved in "outgoing" or "incoming" *pcap* files. These files were then merged by time, converted to a printable format, and fed into a display program.

D.II.A.1. Conversion and formats

The format was designed to capture the header data from the packet, and provide a straightforward representation of the host on which it was captured, the source and destination hosts, and the timestamp provided in the *pcap* file. As an example, an entry looks like:[1]

1425248517.189082 Server Client Client 14:21:57.189082 IP
10.0.0.1.40133 > 10.0.0.2.12345: Flags [S], seq 3737689934, win 29200, options
[mss 1460,sackOK,TS val 255272 ecr 0,nop,wscale 9], length 0

The fields we used were the first five. The first field is the time, the second the source, the third the destination, and the fourth the point of observation. The rest of the line is the text representation of the *pcap* file entry. So this line says that the packet (fifth field on) was observed at time 1425248517.189082 at the client, and the source was the server and the destination was the client.

D.II.A.2. Animation

The demo itself consisted of an animation of the data we captured, because the animation made what was happening clearly visible. In addition to the animation, the actual traces of the data were available in a separate window, so the curious could observe the textual representation of the actual *pcap* (packet) data.

We wrote a simple program in Python, using Pygames, to provide an animated representation of the data. This program used the capture data to produce a green dot representing the packet, and moved it from the source to the destination host via the SDN switch. So, for example, a packet going from host A to host B via the switch would be recorded as outgoing from A (capture point and source) at time t_A and incoming to B (capture point and destination) at time t_B. The animation would show the dot corresponding to the packet leaving A at time t_A and going to the SDN switch. It would then show the dot leaving the SDN switch and arrive at B at time t_B. The timings were critical in order to show the interpacket arrival times accurately.

The idea underlying each scenario is that the controller updates the rules in the switch to launch the attack. Those packets affected by the change were colored red, to emphasize their routing were a result of the attack. This demonstrated our theme of a compromised controller wreaking havoc on the network.

The animation program had several options, including one to print a trace of the packets in a second window (this is why we had the textual representation of the packet in the animation input file), one to pause the animation until the space bar was hit, and one to loop through the data after it had been played once. We used this animation program for both scenarios as discussed below.

One problem is worth noting. Because of some quirks in the retinal display of MacBook Pros, we had to use a special program (SetResX.app) to adjust the screen. This changed the display from putting the animation in the upper left quadrant of a window to filling the window with the animation. This was not a problem on Linux or Windows-based systems.

D.II.A.3. First Scenario: Monitoring or changing packets on a hostile host

In this scenario, traffic is flowing from host A to host B through an SDN switch. The controller, having been compromised, instructs the switch to route all traffic from host A to host B through a third host C. Host C can then read and modify the traffic as desired; if the traffic is encrypted, C can gather statistics that will prove useful in traffic analysis. Host C then sends the packets back to the SDN switch, which (if necessary) rewrites the source address to be that of the user, and sends the packet on to the server. Figure 1 shows this scenario.

[1] The entry in the file is one line, but is broken into 3 lines here for ease of reading.

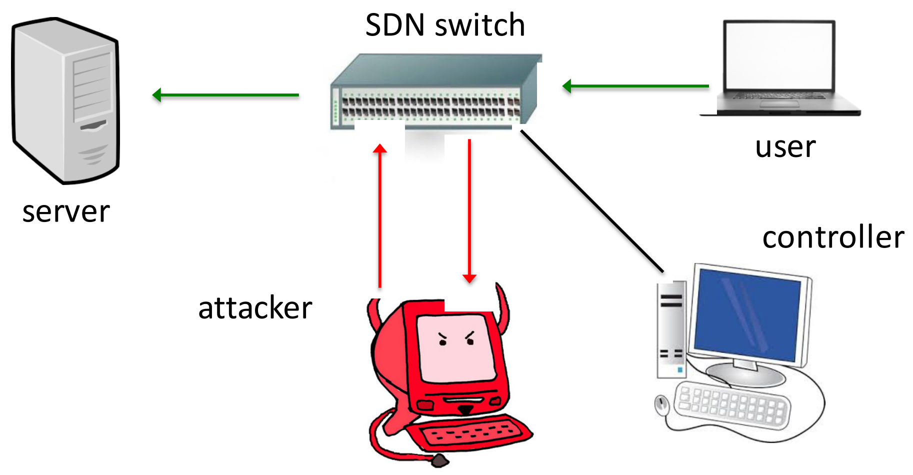

Figure 1. The controller instructs the SDN switch to route traffic between the user and the server through the attacker system.

We set up a network in the Science DMZ testbed to test the effectiveness of this attack. The SDN switch was an HP SDN switch running OpenFlow 1.0, and the other systems (including the controller) all ran Ubuntu Linux. The traffic was diverted and monitored successfully. Figure 2 is a still frame of this animation.

We considered ways to detect this diversion under the assumption that the controller, and through that the SDN switch, were corrupt. The user sees the outgoing packets being properly formed, and the incoming ones as having the source address of the server (because the SDN switch rewrites the source address to be that of the server). Similarly, the server sees the incoming packets as having the source address of the user (because the SDN switch rewrites the source address to be that of the user) and, of course, the outgoing packets are properly formed. The SDN switch can rewrite any of the packet header fields such as time to live (TTL), so none of them can be trusted. But the switch cannot alter time, so the delay introduced by diverting the packets to the attack host C might reveal the compromise. Such a measurement would have to be very accurate and made when the switch was not diverting packets, under a load similar to the one when the packets are being diverted. In short, detection by hosts A or B is unlikely. A monitoring mechanism observing outgoing packets from all hosts and switches could match up the traffic, but most likely such a monitoring mechanism would be driven by data from controllers and switches, so its trustworthiness is questionable.

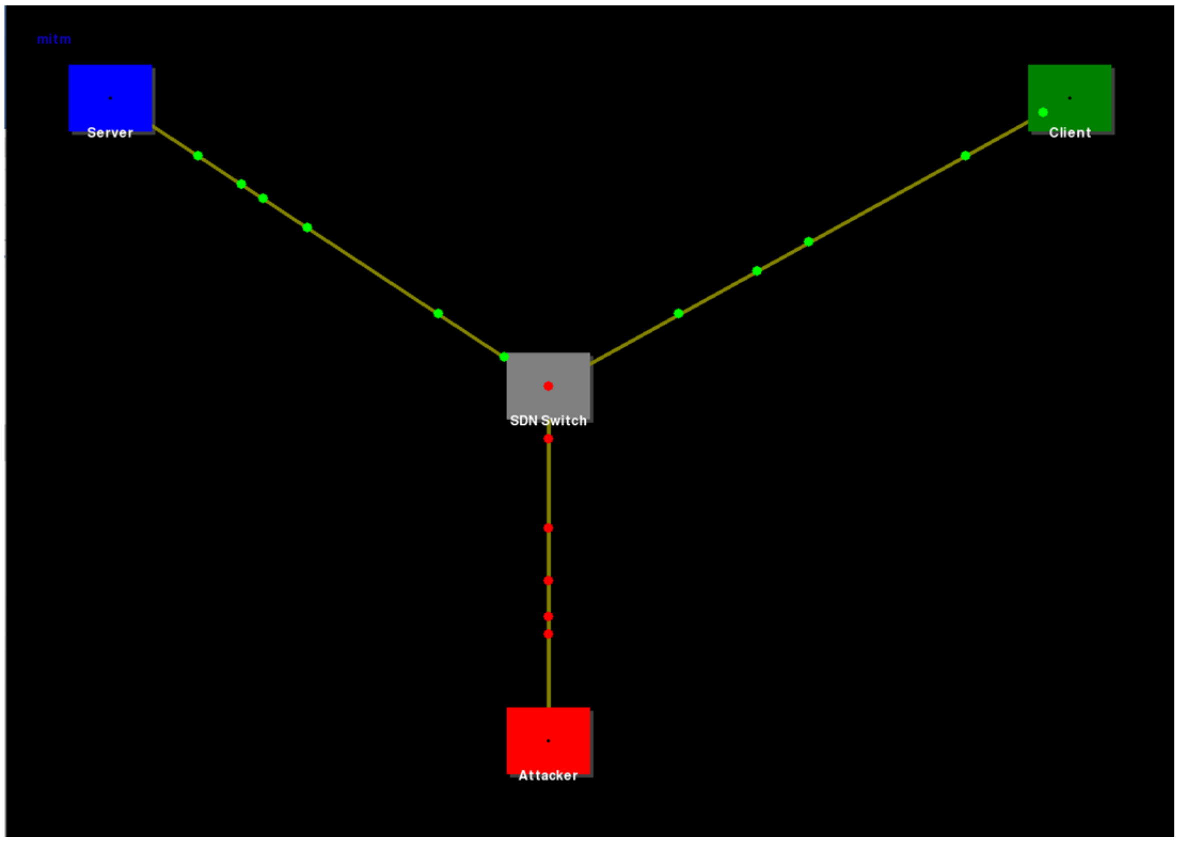

Figure 2. Animation of the first scenario. The green dots represent the normal flow of packets. The red dots represent the rerouting of the packets through the attacker.

D.II.A.4. Second Scenario: Rerouting packets to flood a victim host

The second scenario develops a denial of service attack originating at the SDN switch. The idea is to have the switch copy packets to a target host. If enough packets can be copied to flood the target host, then the switch has effectively launched a denial of service against the host. Figure 3 shows this scenario.

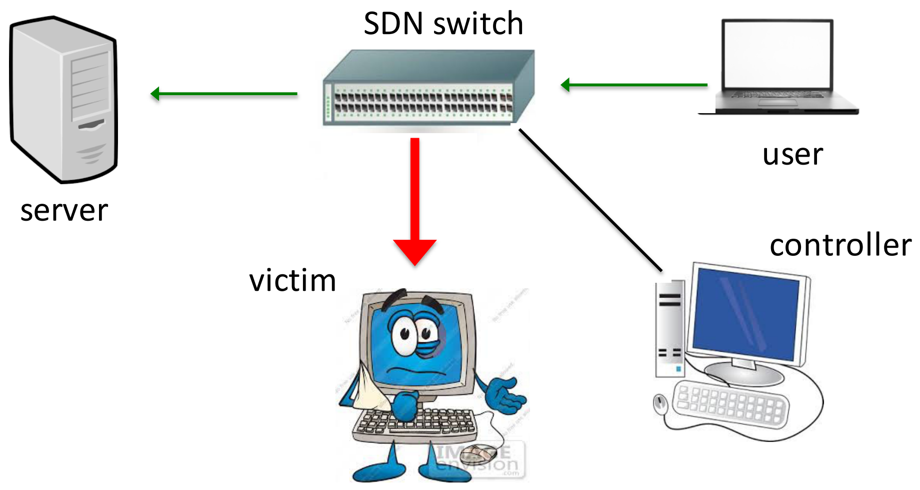

Figure 3. The controller instructs the SDN switch to copy traffic between the user and the server and flood the victim system.

The replication can be done in a variety of ways. For example, the controller can enter a set of group rules (or a single group rule with all the action buckets in that rule to be executed – and add a lot of action buckets) to replicate a packet some very large number of times. Or, if there is much traffic through the switch, copying the packet and forwarding the copy to the victim might suffice as well.

When the victim tries to track down the origin of the attack, the victim will detect it is coming from a host or set of hosts upstream from the SDN switch. The victim is unlikely to see it comes from the switch itself, because network *infrastructure* is rarely seen as a source of attacks. Rather, it is seen as a carrier of attacks from elsewhere. Even if the attacker does determine the origin correctly, it can block the attack only by disconnecting from the network – thereby achieving the attacker's goal of denying service over the network.

For the demonstration for this scenario, we used data collected from a mininet simulation. The setup was the same as for the previous demo: we recorded packets at all three systems, and constructed the input files as before. We then ran our animation program with the configuration set to that of this scenario.

Figure 4 shows a still frame from the early part of the animation in which the switch rules were set to pass packets from the user to the server without diversion.

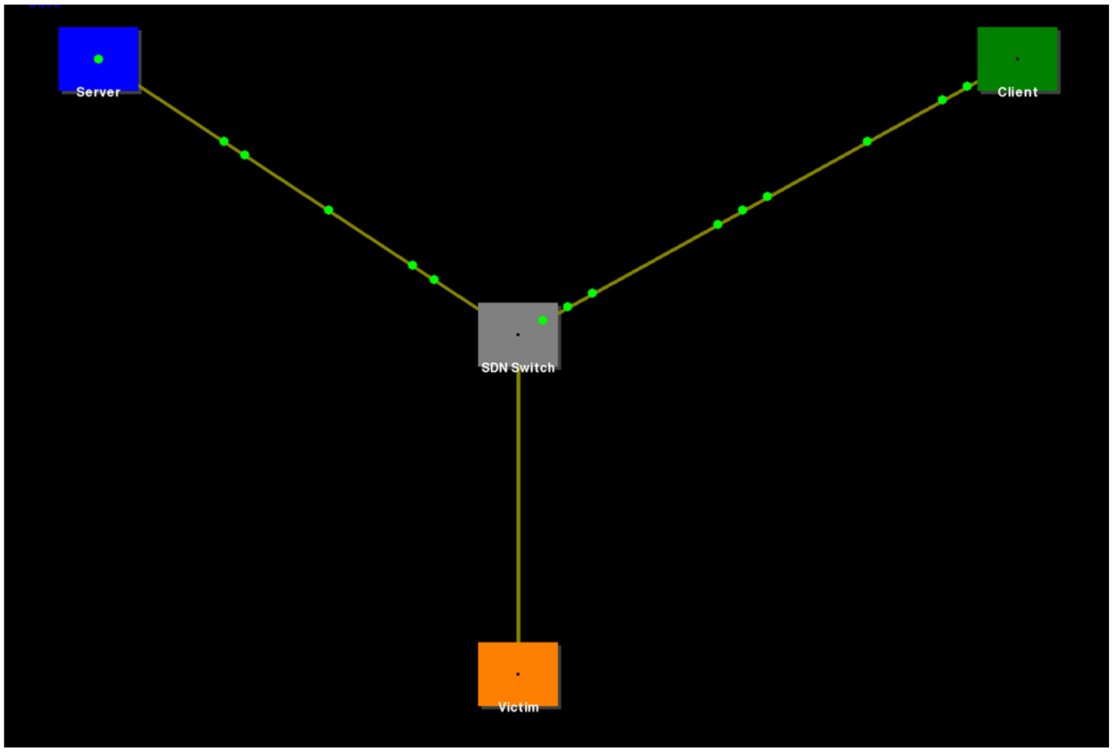

Figure 4. First animation still of the second scenario. All the dots are green because the switch has not yet begun copying packets and sending them to the victim.

After a short time, the controller added group rules to copy packets out to the victim. Multiple rules allow packets to be replicated a number of times. Figure 5 shows a still frame from the animation.

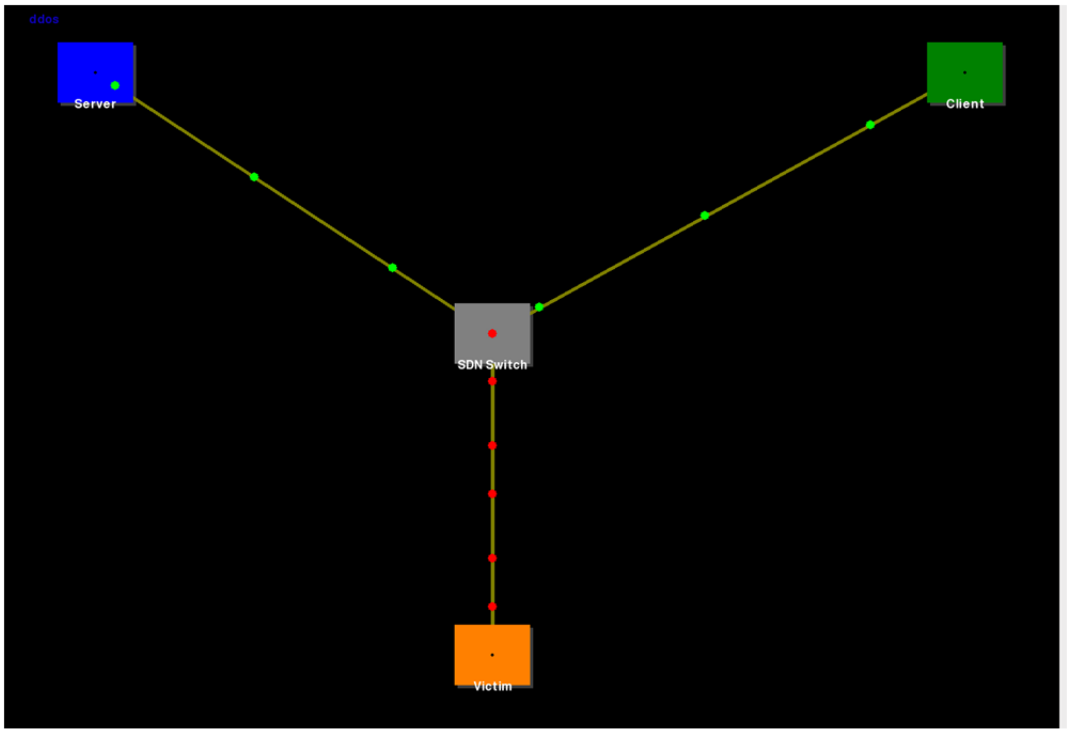

Figure 5. Animation of the second scenario. The green dots represent the normal flow of packets. The red dots represent the rerouting of the packets through the attacker. Note the dot in the center of the switch; the snapshot was made just after the SDN switch generated another packet to send to the victim.

Here, none of the hosts (except, of course, the controller) can determine what the SDN switch is doing with their traffic, as none ever see the replication. Further, if the switch changes the source addresses to random IP addresses, or to those of another network or host, the victim will be deceived as to the origin of the attack. Thus, the origin of the attack (the switch, and ultimately the controller or the program loaded onto the switch) will be unlikely to be identified for a considerable period of time.

D.II.A.5. Conclusion

At GEC 22, we showed and successfully demonstrated both scenarios described above. Due to the limited amount of resources we could not run the two demos in parallel. Therefore, we decided to run primarily the first demo demonstrating the diversion and reading (and, possibly, modifying) traffic between the servers, while running the second demo based on the specific requests and interest of the demo session attendees.

B. Project participants

Prof. S. J. Ben Yoo	*Heterogeneous Multi-Domain Network Testbed*	UC Davis, PI
Prof. Matt Bishop	*Security in Scalable Programmable Networks*	UC Davis, Co-PI
Prof. Chen-Nee Chuah	*Monitoring in Scalable Software Defined Networks*	UC Davis, Co-PI
Mr. Brian Perry	*Scenario development, data collection from mininet*	UC Davis
Mr. Abhishek Gupta	*Data collection from SDN network*	UC Davis
Mr. Steven Templeton	*GUI, replay software for GEC22 demo*	UC Davis

C. Publications (individual and organizational)

[1] L. Lei, P. Wei-Ren, R. Casellas, T. Tsuritani, I. Morita, R. Martinez, *et al.*, "Dynamic OpenFlow-Based Lightpath Restoration in Elastic Optical Networks on the GENI Testbed," *Lightwave Technology, Journal of,* vol. 33, pp. 1531-1539, 2015.

[2] L. Liu, Z. Zhu, X. Wang, G. Song, C. Chen, X. Chen, *et al.*, "Field Trial of Broker-based Multi-domain Software-Defined Heterogeneous Wireline-Wireless-Optical Networks," in *Optical Fiber Communication Conference*, Los Angeles, California, 2015, p. Th3J.5.

D. Outreach activities

N/A

E. Collaborations

N/A

F. Other Contributions and Future Plans

Our next step is to examine attacks on the infrastructure. For example what happens if the switch is set to forward packets that do not satisfy any rules to the controller, and a malicious host floods the switch with such packets? Can the link between the switch and the controller be disrupted or blocked? We plan to do this over the next few months, both on the mininet simulation and on the Science DMZ testbed.