Title : SDN-based Traffic Analysis Resistant Network (TARN) Architecture

Experiment Overview

In this experiment, we will construct a prototype of SDN-based traffic analysis resistant network architecture (TARN), with the support of both the GENI and PEERING testbeds. The architecture prototype binds a across-Internet end-to-end communication session to randomized, short-lived, perpetually changing server IP addresses. In this experiment, we will use GENI testbed to provide network resources at the network edge; while the PEERING testbed allows to announce BGP prefixes to the Internet and do BGP routing at the network core.

This experiment will take approximately 30 minutes to setup, and another 30 minutes to test.

Background

Traffic analysis refers to methods that discover end-to-end Internet communications patterns by observing side-channels. This works even when the traffic is encrypted. Today's Internet Protocol (IP) networks are vulnerable to traffic analysis; which makes Internet censorship, man-in-the-middle (MITM) attacks and network surveillance possible. The presence of clear-text source and destination IP addresses in TCP/IP headers makes communications flows trivially easy for adversaries to detect and attack. To make traffic analysis more difficult, proxy-networks (Tor [1], Psiphon [2], Lanten [3], VPNs...) obscure destination IP addresses by routing traffic through one or more proxy nodes. However, proxy nodes are still being detected and blocked by nation states, among others. In addition, such solutions force users to trust intermediate proxy nodes, which sometimes execute MITM attacks.

A software defined networking (SDN) based solution called TARN has been proposed to provide an end-to-end network architecture to remove the basic traffic analysis vulnerability. To resist traffic analysis, TARN binds communication sessions to randomized, short-lived, perpetually changing IPv4/IPv6 addresses. This traffic analysis resistance is achieved through the placement of a SDN at the network edge, combined with the use of BGP routing in the network's core.

Experiment Setup and Run

Experiment Overview

Since the resistance is achieved through the placement of a SDN at the network edge, combined with the use of BGP routing in the network core. The GENI experiment we designed to simulate this is then shown in Figure 1

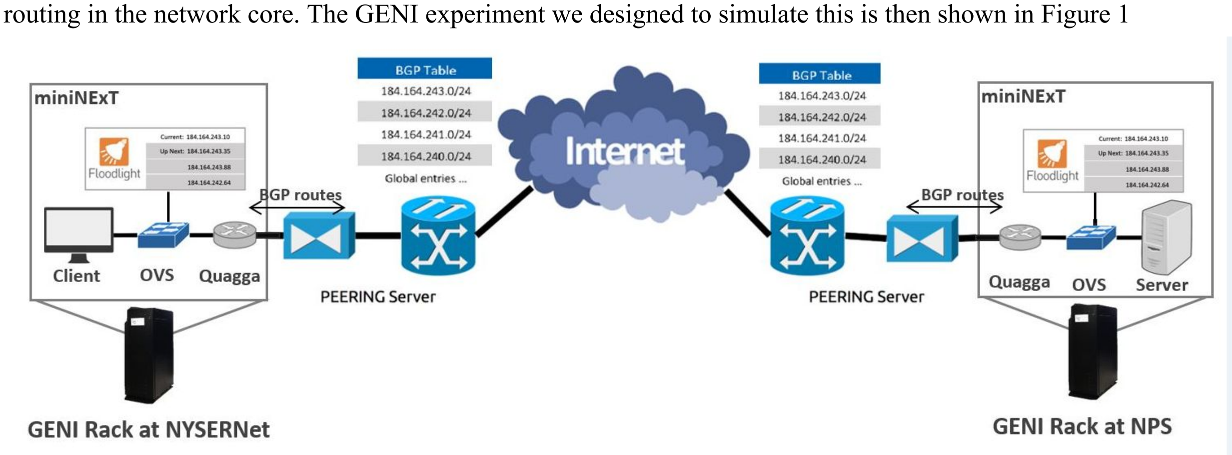


Figure 1. GENI Experiment Design

There are two reserved GENI racks that represents the server and client side, located at NPS and NYSERNet, respectively. On each GENI rack, there is a virtual network emulator called miniNExT [4] installed. Inside each miniNExT, it includes a Floodlight manager that rewrite packet header, an OVS that interacts with Floodlight, and a Quagga router [5] running as a virtual BGP router.

Experiment Setup

1) Reserve GENI resources

In the GENI portal, create a new slice and then click "Add Resources". Select two GENI VMs from different GENI sites. In this experiment, GENI VMs is working as a ingress point that connects to the Internet.

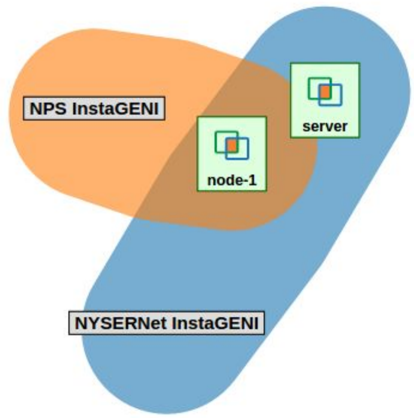


Figure 2 : Experiment GENI Topology

2) Install software on GENI node

On each GENI node, the following software needs to be installed:

1. Open vSwitch 2.3.1 – http://docs.openvswitch.org/en/latest/intro/install/
2. Mininet 2.1.0p2 – https://github.com/mininet/mininet/blob/2.1.0p2/INSTALL
3. MiniNExT 1.4.0 – https://github.com/USC-NSL/miniNExT
4. Quagga – http://www.nongnu.org/quagga/docs/docs-info.html#Installation
5. EAGERProject – https://github.com/OpenFlow-Clemson/EAGERProject/tree/develop
6. EAGERFloodlight – https://github.com/cbarrin/EAGERFloodlight/tree/develop

Floodlight needs Java 8 in order to run and must be compiled using either Ant or Maven.

3) Set up PEERING testbed.

To reproduce this experiment, you will need to have an account for both the GENI and PEERING testbeds. For GENI, you will need to join a project to reserve the nation-wide distributed network resources. You will also need to generate a SSH key in order to log into and use the reserved network resources. In the PEERING testbed case, you will need to write a proposal to the administrator to request BGP prefixes and describe the use of those allocated BGP prefixes. After the proposal is approved, you will be assigned a list of IP prefixes to use, along with PEERING-issued certificates. The following link explains how to setup OpenVPN tunnels using these certificates: https://github.com/PEERINGTestbed/client#peering-account-setup

4) Attach miniNExT on PEERING

In EAGER project github, https://github.com/cbarrin/EAGERFloodlight/tree/develop

- Change the IP on start.py to the assigned OpenVPN IP assigned by PEERING testbed

- Change the IP on bgp config file to the assigned OpenVPN IP assigned by PEERING testbed

5) Running Floodlight, using REST API to enable randomized module
Floodlight must have the Randomizer module configured and enabled in order for TARN to work properly. The easiest way to do this is to modify Floodlight's properties file, located here:
https://github.com/cbarrin/EAGERFloodlight/blob/develop/src/main/resources/floodlightdefault.properties
Under the Randomizer group, there are four important fields that must be set correctly – enabled, randomize, lanport, and wanport. The 'enabled' field will need to be set to *true* for each Floodlight instance. The 'randomize' field will only be set to *true* for the Floodlight instance who will have a randomized host behind it. The 'lanport' and 'wanport' fields will be the OVS port numbers of the host–connected and Quagga–connected ports, respectively; they are often ports 1 and 2.

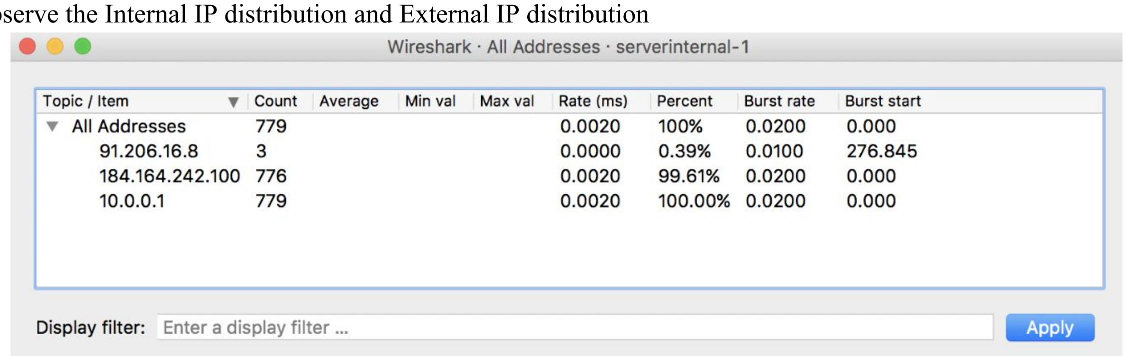
Once the properties file is set, Floodlight may be recompiled and run. The Randomizer should start automatically, recognizing any packets destined for a randomized host and inserting rewrite flows accordingly.

Experiment Evaluation

Evaluation Objective

The main objective for this experiment is to setup and evaluate the proposed SDN-based TARN architecture using GENI testbed. At this point, the evaluation mainly focuses on the host-to-host communication with the condition that the host external IP addresses keep changing. Specifically, the architecture will be evaluated for feasibility of
1) Randomized IP and Session Maintenance via SDN at network edge
2) BGP routing at network core.

Evaluation on (1)
* end-to-end ping test and use tcpdump to capture a series of packets on the network edge
* observe the Internal IP distribution and External IP distribution

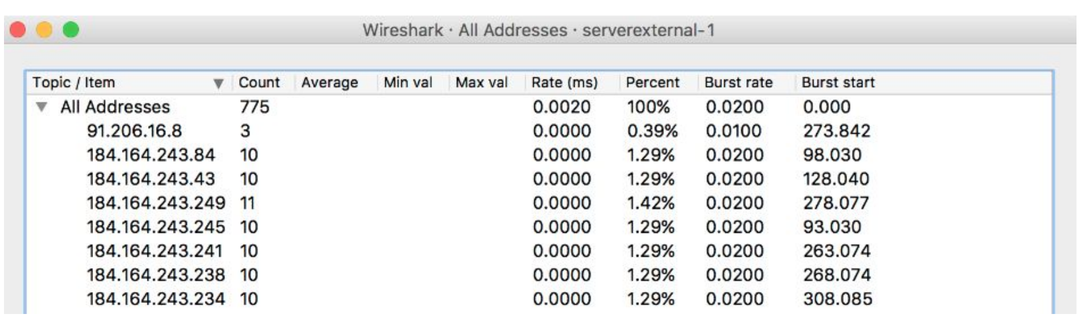


Wireshark · All Addresses · serverinternal-1

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
All Addresses	779				0.0020	100%	0.0200	0.000
91.206.16.8	3				0.0000	0.39%	0.0100	276.845
184.164.242.100	776				0.0020	99.61%	0.0200	0.000
10.0.0.1	779				0.0020	100.00%	0.0200	0.000

Display filter: Enter a display filter ... Apply

Figure 3. Internal IP distribution from host perspective



Wireshark · All Addresses · serverexternal-1

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
All Addresses	775				0.0020	100%	0.0200	0.000
91.206.16.8	3				0.0000	0.39%	0.0100	273.842
184.164.243.84	10				0.0000	1.29%	0.0200	98.030
184.164.243.43	10				0.0000	1.29%	0.0200	128.040
184.164.243.249	11				0.0000	1.42%	0.0200	278.077
184.164.243.245	10				0.0000	1.29%	0.0200	93.030
184.164.243.241	10				0.0000	1.29%	0.0200	263.074
184.164.243.238	10				0.0000	1.29%	0.0200	268.074
184.164.243.234	10				0.0000	1.29%	0.0200	308.085

Figure 4. External IP distribution from Internet perspective

From each host perspective, every packet header in the session should only contain the internal IP address, since each host only knows the real source IP address and the real destination IP address. After packet go across network edge, the header will be rewritten. Therefore, from the outside world perspective, the internal host IP address will never been observed but instead of many randomized external IP addresses.

Evaluation on (2)

* Confirm BGP routes are propagated to the Internet
   -- Log into public AT&T router server : telnet route-server.ip.att.net
   -- Type "show route 184.164.243.0/24" and "show route 184.164.242.0/24"
   -- Result shows below, which proves the BGP routes is on Internet

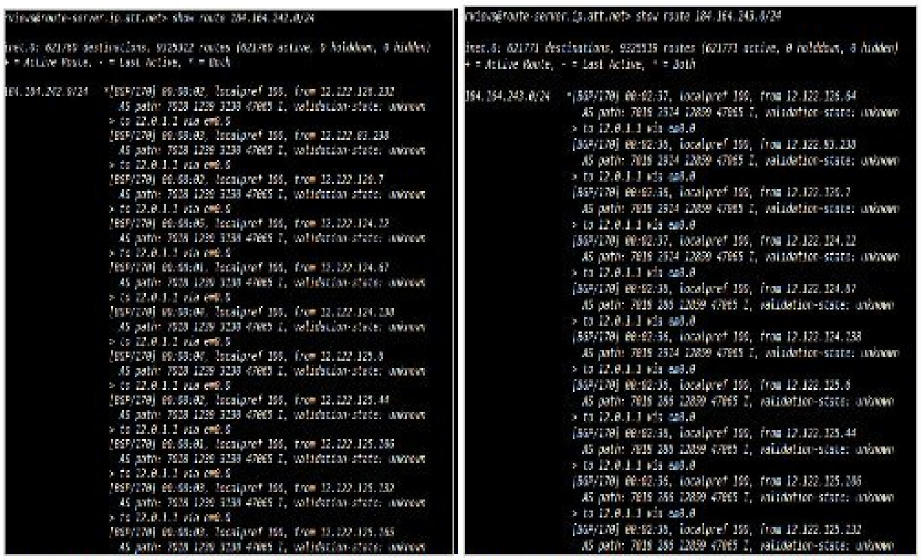


Figure 5. Advertised BGP routes 184.164.243.0/24 and 184.164.242.0/24

* Confirm BGP routes at local Quagga router
   -- Inside miniNExT util directory, do "./mx <host_name> vtysh", this will allow you to log into virtual Quagga router
   -- Do "sh ip bgp nei <neighbor_IP> advertised-route", this shows you the advertised routes from local perspective

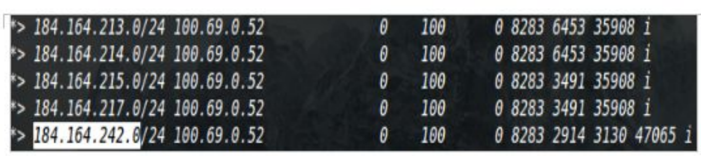


Figure 6. Server side Quagga received BGP prefix 184.164.242.0/24 announced from client side

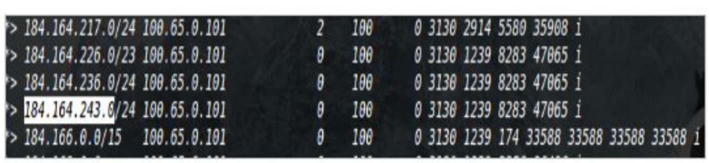


Figure 7. Client side Quagga received BGP prefix 184.164.243.0/24 announced from server side

   -- Now, you have confidence that the BGP routes is setting up correctly