Google Cloud Packet Mirroring with OpenSource IDS

GSP474

Google Cloud Self-Paced Labs

Overview

Traffic Mirroring is a key feature in Google Cloud networking for security and network analysis. Its functionality is similar to that of a network tap or a span session in traditional networking. In short, Packet Mirroring captures network traffic (ingress and egress) from select "mirrored sources", copies the traffic, and forwards the copy to "collectors". It is important to note that Packet Mirroring captures the full payload of each packet and thus consumes additional bandwidth. Because Packet Mirroring is not based on any sampling period, it is able to be used for better troubleshooting, security solutions, and higher layer application based analysis.

Packet Mirroring is founded on a "Packet Mirroring Policy", which contains the following attributes:

- Region
- VPC Network(s)
- Mirrored Source(s)
- Collector (destination)
- Mirrored traffic (filter)

Here are a some key points that also need to be considered:

- Only TCP, UDP and ICMP traffic may be mirrored. This, however, should satisfy the majority of use cases.
- "Mirrored Sources" and "Collectors" must be in the SAME Region, but can be in different zones and even different VPCs, as long as those VPCs are properly Peered.
- Additional bandwidth charges apply, especially between zones. To limit the traffic being mirrored, filters can be used.

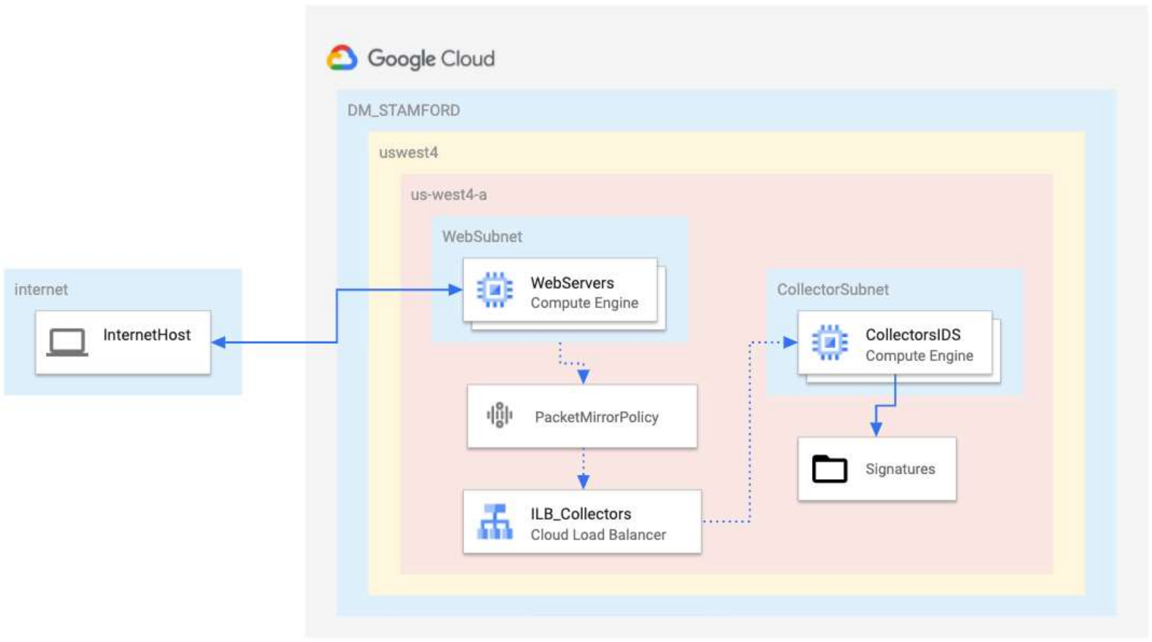
One prime use case for "Packet Mirroring" is to use it in an Intrusion Detection System (IDS) solution. Some cloud-based IDS solutions require a special service to run on each source VM, or to put an IDS virtual appliance in-line between the network source and destination. Both of these have significant implications. For example, the service based solution, though fully distributed, requires that the guest operating system supports the software. The "in-line" solution can create a network bottleneck as all traffic must be funneled through the IDS appliance. The in-line solution will also not be able to capture "east-west" traffic within VMs in the same VPC.

Google Cloud Packet Mirroring does not require any additional software on the VMs and it is fully distributed across each of the mirrored virtual machines. The "Collector" IDS is placed out-of-path using an Internal Network Load Balancer (ILB) and will receive both "north-south" traffic and "east-west" traffic.

Packet Mirroring lab description

To demonstrate how Packet Mirroring can be used with an IDS consider this example using OpenSource IDS Suricata.

- A single VPC with 2 subnets, one for mirrored sources and one for the collector
- 2 Web servers created with a public IP address
- 1 Collector server (IDS) created with NO public IP for security reasons
- CloudNAT enabled for Internet access as needed
- All VMs created in the same region and zone, for simplicity and cost reasons
In this lab you will create a Google Cloud environment, configure the "Collector" ILB, configure the Packet Mirror Policy, as well as install and configure [Suricata] (https://suricata-ids.org/) on a virtual instance to act as an IDS. Once complete, network tests will be performed to validate the configuration and use of Packet Mirroring with the Open Source IDS. A very stripped down rule-set and Suricata configuration is used to simplify the demonstration..



Objectives:

- Build out a Google Cloud Networking environment as shown in the diagram above
- Create 2 virtual machines with `gcloud` commands to act as WEB SERVERS
- Create a single virtual machine with `gcloud` commands to act as IDS
- Create an Internal LoadBalancer (ILB) to act as a "collector" for Packet Mirroring
- Install and configure an Open Source IDS (Suricata) on the IDS VM

- Review some basic IDS alert rules
- Create a Packet Mirror Policy
- Test Packet Mirroring by generating network traffic to the "mirrored" subnet
- Test Suricata IDS by generating network traffic to simulate an IDS event and review IDS logging

Setup and requirements

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

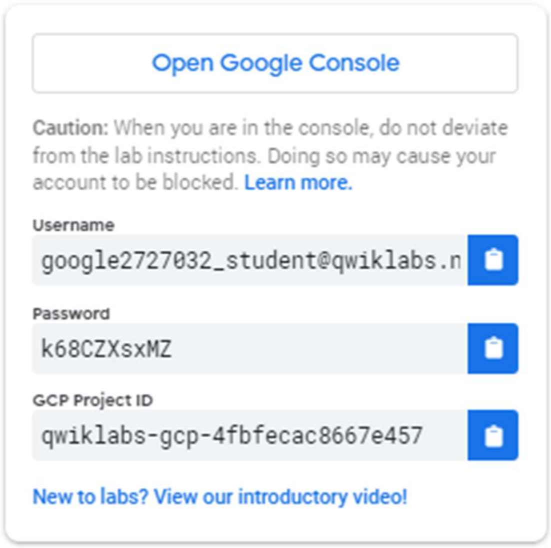
To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.
  Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

Note: If you are using a Pixelbook, open an Incognito window to run this lab.

How to start your lab and sign in to the Google Cloud Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Username

google2727032_student@qwiklabs.n

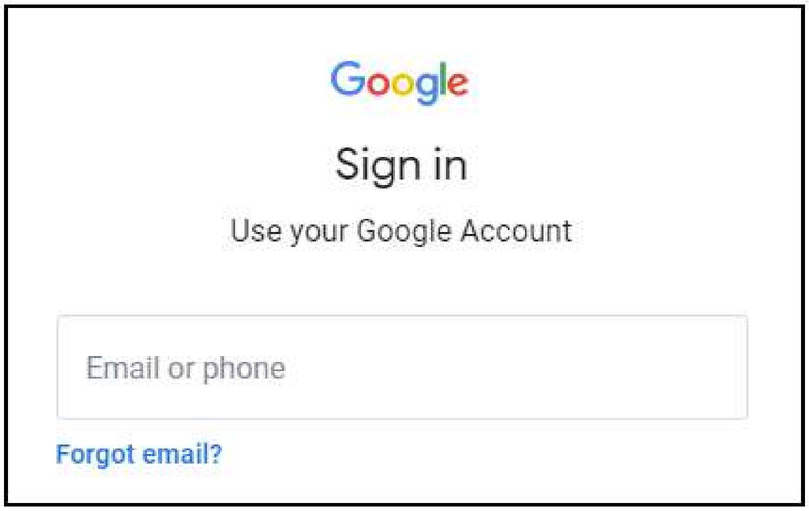
Password

k68CZXsxMZ

GCP Project ID

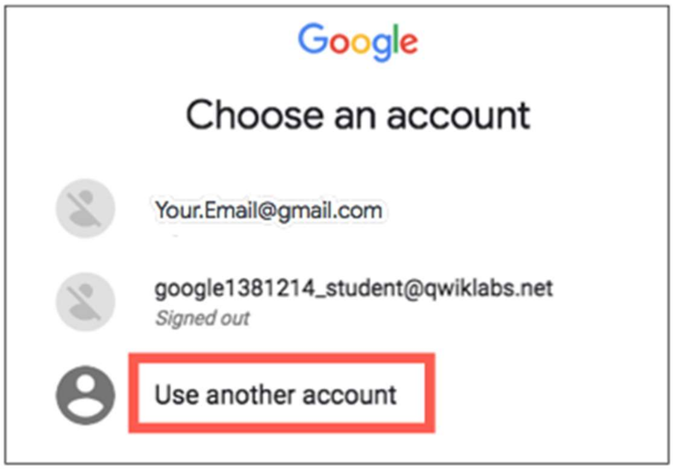
qwiklabs-gcp-4fbfecac8667e457

2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



Tip: Open the tabs in separate windows, side-by-side.

If you see the **Choose an account** page, click **Use Another Account**.



3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

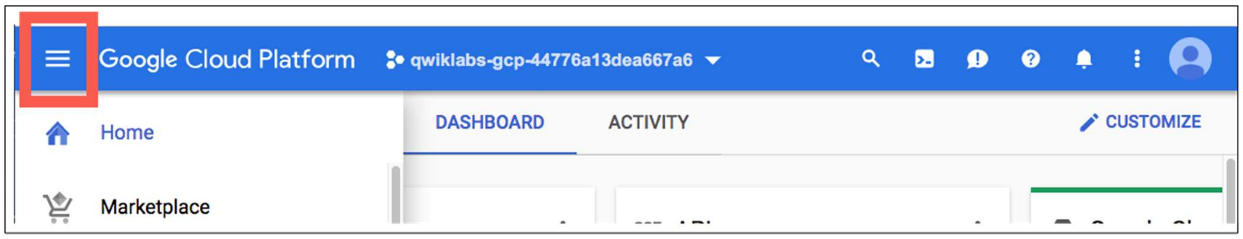
   Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

   - Accept the terms and conditions.
   - Do not add recovery options or two-factor authentication (because this is a temporary account).
   - Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

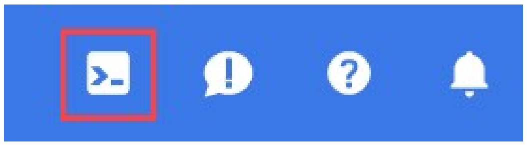
Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.


Google Cloud Platform
qwiklabs-gcp-44776a13dea667a6
Home
Marketplace
DASHBOARD
ACTIVITY
CUSTOMIZE

Build a networking footprint

In this section, you will create a VPC and create 2 subnets inside that VPC. This will all be done using `gcloud` CLI commands inside Google Cloud Shell.

If not yet done, open the **Google Cloud Shell** by clicking on the icon on the top right of the screen:



Run the following to create a virtual private network:

```
gcloud compute networks create dm-stamford \
--subnet-mode=custom
```

Add a subnet to the VPC for mirrored traffic in us-west4:

```
gcloud compute networks subnets create dm-stamford-uswest4 \
--range=172.21.0.0/24 \
--network=dm-stamford \
--region=us-west4
```

Add a subnet to the VPC for the collector in us-west4:

```
gcloud compute networks subnets create dm-stamford-uswest4-ids \
--range=172.21.1.0/24 \
--network=dm-stamford \
--region=us-west4
```

Click *Check my progress* to verify the objective.

Create firewall rules and Cloud NAT

A total of three firewall rules will be required to complete this lab.

- Rule 1 allows the standard http port (TCP 80) and the ICMP protocol to all VMs from all sources.
- Rule 2 allows the IDS to receive ALL traffic from ALL sources. Be careful NOT to give the IDS VM a public IP in the later sections.
- Rule 3 allows the "GCP IAP Proxy" IP range TCP port 22 to ALL VMs, enabling you to ssh into the VMs via the Cloud Console
  Run the following commands to create the firewall rules:

```
gcloud compute firewall-rules create fw-dm-stamford-allow-any-web \
--direction=INGRESS \
--priority=1000 \
--network=dm-stamford \
--action=ALLOW \
--rules=tcp:80,icmp \
--source-ranges=0.0.0.0/0

gcloud compute firewall-rules create fw-dm-stamford-ids-any-any \
--direction=INGRESS \
--priority=1000 \
--network=dm-stamford \
--action=ALLOW \
--rules=all \
--source-ranges=0.0.0.0/0 \
--target-tags=ids

gcloud compute firewall-rules create fw-dm-stamford-iapproxy \
--direction=INGRESS \
--priority=1000 \
--network=dm-stamford \
--action=ALLOW \
--rules=tcp:22,icmp \
--source-ranges=35.235.240.0/20
```

Click *Check my progress* to verify the objective.

Create a CloudRouter

As a prerequisite for Cloud NAT, a Cloud Router must first be configured in the respective region:

```
gcloud compute routers create router-stamford-nat-west4 \
--region=us-west4 \
--network=dm-stamford
```

Configure a Cloud NAT

To provide Internet access to VMs without a public IP, a Cloud NAT must be created in the respective region:

```
gcloud compute routers nats create nat-gw-dm-stamford-west4 \
--router=router-stamford-nat-west4 \
--router-region=us-west4 \
--auto-allocate-nat-external-ips \
--nat-all-subnet-ip-ranges
```

The IDS VM will be created without a public IP to make sure that it is inaccessible from the Internet. However, it will need internet access to download updates and install Suricata packages.

Click *Check my progress* to verify the objective.

Create Virtual Machines

Create an Instance Template for a WebServer

This template prepares an Ubuntu server in `us-west4` and installs a simple web service:

```
gcloud compute instance-templates create template-dm-stamford-web-us-west4 \
--region=us-west4 \
--network=dm-stamford \
--subnet=dm-stamford-uswest4 \
--machine-type=g1-small \
--image=ubuntu-1604-xenial-v20200807 \
--image-project=ubuntu-os-cloud \
--tags=webserver \
--metadata=startup-script='#! /bin/bash
  apt-get update
  apt-get install apache2 -y
  vm_hostname="$(curl -H "Metadata-Flavor:Google" \
  http://169.254.169.254/computeMetadata/v1/instance/name)"
  echo "Page served from: $vm_hostname" | \
  tee /var/www/html/index.html
  systemctl restart apache2'
```

Create a Managed Instance Group for the WebServers

This command uses the Instance Template in the previous step to create 2 web servers:

```
gcloud compute instance-groups managed create mig-dm-stamford-web-uswest4 \
    --template=template-dm-stamford-web-us-west4 \
    --size=2 \
    --zone=us-west4-a
```

Create an Instance Template for the IDS VM

This template prepares an Ubuntu server in us-west4 and with no public IP:

```
gcloud compute instance-templates create template-dm-stamford-ids-us-west4 \
--region=us-west4 \
--network=dm-stamford \
--no-address \
--subnet=dm-stamford-uswest4-ids \
--image=ubuntu-1604-xenial-v20200807 \
--image-project=ubuntu-os-cloud \
--tags=ids,webserver \
--metadata=startup-script='#! /bin/bash
  apt-get update
  apt-get install apache2 -y
  vm_hostname="$(curl -H "Metadata-Flavor:Google" \
  http://169.254.169.254/computeMetadata/v1/instance/name)"
  echo "Page served from: $vm_hostname" | \
  tee /var/www/html/index.html
  systemctl restart apache2'
```

Create a Managed Instance Group for the IDS VM

This command uses the Instance Template in the previous step to create 1 VM to be configured as our IDS. The installation of Suricata will be addressed in a later section.

```
gcloud compute instance-groups managed create mig-dm-stamford-ids-uswest4 \
    --template=template-dm-stamford-ids-us-west4 \
    --size=1 \
    --zone=us-west4-a
```

Click *Check my progress* to verify the objective.

Create Internal LoadBalancer

Packet Mirroring uses an internal load balancer (ILB) to forward mirrored traffic to a group of collectors. In this case, the collector group contains a single VM.

Create a basic health check for the backend services:

```
gcloud compute health-checks create tcp hc-tcp-80 --port 80
```

Create a backend service group to be used for an ILB:

```
gcloud compute backend-services create be-dm-stamford-suricata-us-west4 \
--load-balancing-scheme=INTERNAL \
--health-checks=hc-tcp-80 \
--network=dm-stamford \
--protocol=TCP \
--region=us-west4
```

Add the created IDS managed instance group into the backend service group created in the previous step:

```
gcloud compute backend-services add-backend be-dm-stamford-suricata-us-west4 \
--instance-group=mig-dm-stamford-ids-uswest4 \
--instance-group-zone=us-west4-a \
--region=us-west4
```

Create a front end forwarding rule to act as the collection endpoint:

```
 gcloud compute forwarding-rules create ilb-dm-stamford-suricata-ilb-us-west4 \
 --load-balancing-scheme=INTERNAL \
 --backend-service be-dm-stamford-suricata-us-west4 \
 --is-mirroring-collector \
 --network=dm-stamford \
 --region=us-west4 \
 --subnet=dm-stamford-uswest4-ids \
 --ip-protocol=TCP \
 --ports=all
```

Note:that even though TCP is listed as the protocol, the actual type of mirrored traffic will be configured in the Packet Mirror Policy in a future section. Also, note the "`--is-mirroring-collector`" flag is set.

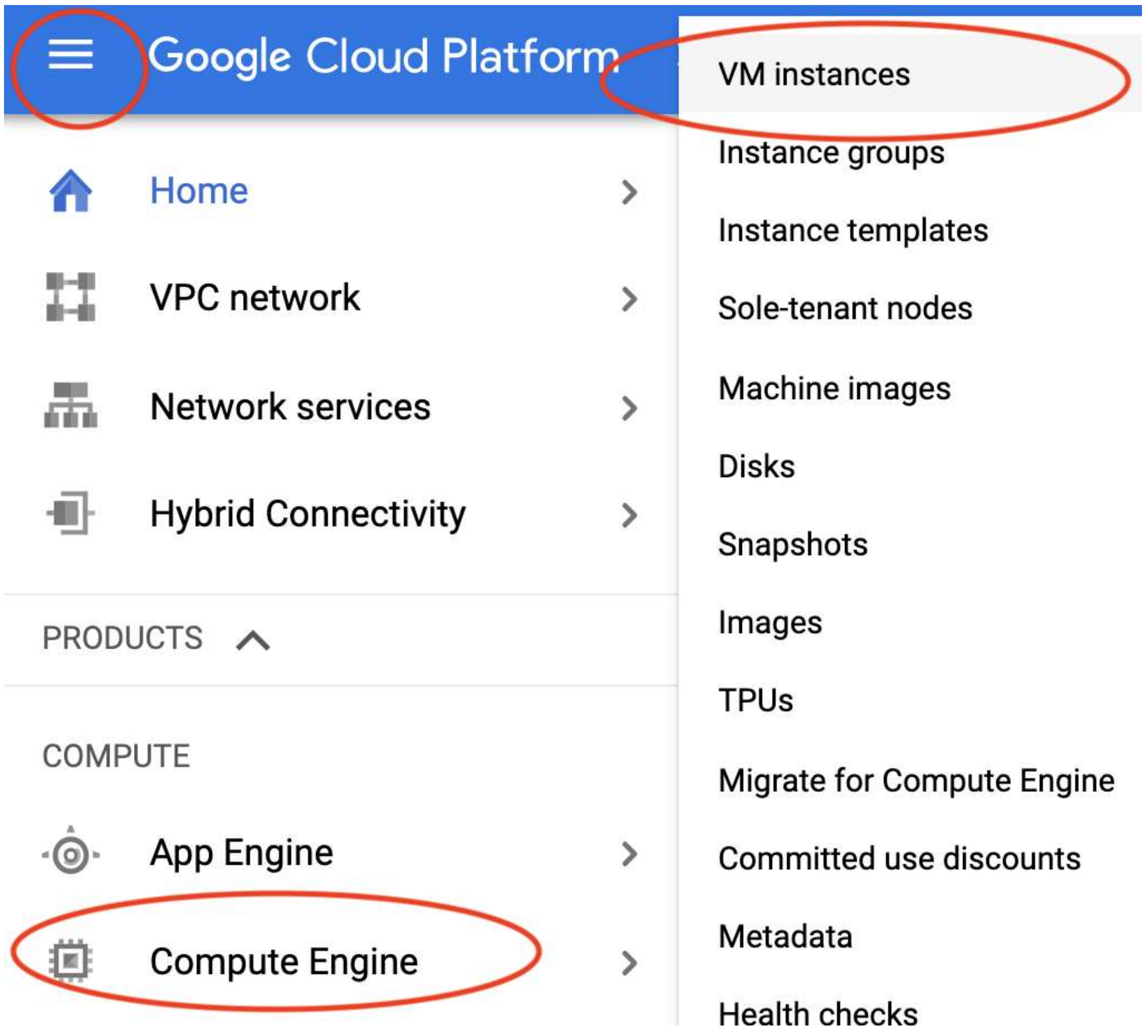
Click *Check my progress* to verify the objective.

Install Open Source IDS - Suricata

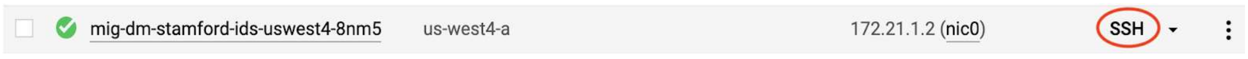
Note: For the next 2 sections, you will need to ssh into the IDS VM and run the commands in its shell. Be careful not to run the commands in the Cloud Shell.

SSH into the IDS VM

Using the Cloud Console, from the **Navigation menu**, navigate to **Compute Engine** > **VM Instances**.



Click on the **SSH** button of your IDS VM.



This opens a new window allowing you to run commands inside the IDS VM.

Update the IDS VM:

```
sudo apt-get update -y
```

Install Suricata dependencies:

```
sudo apt-get install libpcre3-dbg libpcre3-dev autoconf automake libtool libpcap-dev
libnet1-dev libyaml-dev zlib1g-dev libcap-ng-dev libmagic-dev libjansson-dev
libjansson4 -y

sudo apt-get install libnspr4-dev -y

sudo apt-get install libnss3-dev -y

sudo apt-get install liblz4-dev -y

sudo apt install rustc cargo -y
```

Install Suricata:

```
sudo add-apt-repository ppa:oisf/suricata-stable -y

sudo apt-get update -y

sudo apt-get install suricata -y
```

If you receive an error during installation, please proceed with the following verification step.

Verify Installation

Use the following command to verify the installation and check the installed version of Suricata:

```
suricata -V
```

The output should be something similar to this:

```
This is Suricata version 5.0.3 RELEASE
```

Configure and review Suricata

Commands and steps for the following section should also be done inside the SSH of IDS/Suricata VM.

Stop the Suricata service and backup the default configuration file:

```
sudo systemctl stop suricata

sudo cp /etc/suricata/suricata.yaml /etc/suricata/suricata.backup
```

Download and replace new Suricata configuration file and abridged rules file

The new configuration file updates the collector interface as well as only alerts on a very small set of traffic as configured in the `my.rules` file and the `suricata.yaml` file. The defaults for Suricata rulesets and alerting are quite extensive and superfluous for this lab, so run the following commands to copy the files.

```
wget https://storage.googleapis.com/tech-academy-enablement/GCP-Packet-Mirroring-with-OpenSource-IDS/suricata.yaml

wget https://storage.googleapis.com/tech-academy-enablement/GCP-Packet-Mirroring-with-OpenSource-IDS/my.rules

sudo mkdir /etc/suricata/poc-rules

sudo cp my.rules /etc/suricata/poc-rules/my.rules

sudo cp suricata.yaml /etc/suricata/suricata.yaml
```

Start the Suricata service

Occasionally, the service needs to be restarted. The `restart` command is included in this step to account for that possibility:

```
sudo systemctl start suricata

sudo systemctl restart suricata
```

Review Simple Suricata rules for testing

For this lab, the ruleset for Suricata had been trimmed down to 4. However, the default Suricata installation has a vast and extensive ruleset. Condensing the alerting down to such a succinct and simple list is better for this lab exercise so that each one can be easily tested.

```
cat /etc/suricata/poc-rules/my.rules
```

Output should show a total of 4 rules and a description of each.

```
####RULES#####
#UDP ALERTS
alert udp $HOME_NET any -> 8.8.8.8 53 (msg:"BAD UDP DNS REQUEST"; sid:99996; rev:1;)

#HTTP ALERTS
alert http any any -> $HOME_NET 80 (msg:"BAD HTTP PHP REQUEST"; http.uri;
content:"index.php"; sid:99997; rev:1;)

#ICMP ALERTS
alert icmp any any -> $HOME_NET any (msg:"BAD ICMP"; sid:99998; rev:1;)

#TCP ALERTS
alert tcp $HOME_NET any -> any 6667 (msg:"BAD TCP 6667 REQUEST"; sid:99999; rev:1;)
```

Note: The standard rules files may typically be located in /etc/suricata/rules/ or /var/lib/suricata/rules. It was reconfigured for a different location for this lab in Step 2.

Configure Packet Mirror Policy

For this section of the lab return to Cloud Shell

Setting up the Packet Mirror policy can be completed in one simple command (or by going through a "wizard" in the GUI). In this command, you specify all 5 attributes mentioned in the Packet Mirroring Description section.

- Region
- VPC Network(s)
- Mirrored Source(s)
- Collector (destination)
- Mirrored traffic (filter)

You may notice that there is no mention of "mirrored traffic". This is because the policy will be configured to mirror "ALL" traffic and no filter is needed. The policy will mirror both ingress and egress traffic and forward it to the Suricata IDS device, which is part of the collector ILB.

Configure Packet Mirroring policy:

```
gcloud compute packet-mirrorings create mirror-dm-stamford-web \
--collector-ilb=ilb-dm-stamford-suricata-ilb-us-west4 \
--network=dm-stamford \
--mirrored-subnets=dm-stamford-uswest4 \
--region=us-west4
```

At this point, the Packet Mirroring and the Suricata configuration should be complete. The following sections will test both.

Click *Check my progress* to verify the objective.

Test Packet Mirroring

For the next section, you will need to access the **IDS VM shell**. If the shell window is still open, use that. If the shell window has been closed, reconnect.

You will also be using **Cloud Shell** to act as the "Internet Client".

Take a few minutes to find the External IP of both WEB VMs. In the Cloud Console, navigate to **Compute Engine > VM Instances** and note down **External IP** of both WEB VMs. These will be referred to as **[PUBLIC_IP_WEB1]** and **[PUBLIC_IP_WEB2]**, respectively.

You can also gather the same information through `gcloud` commands from Cloud Shell:

```
gcloud compute instances list
```

Return to the IDS VM shell

Test Packet Mirroring

Run a packet capture (tcpdump) on the IDS/Suricata VM with the following filters:

```
sudo tcpdump -i ens4 -nn -n "(icmp or port 80) and net 172.21.0.0/24"
```

Note: The 172.21.0.0/24 network is the MIRRORED subnet; and the IDS VM is not part of that subnet. If Packet Mirroring is correctly configured, the IDS VM should receive mirrored traffic for that subnet.

Generate traffic to the "mirrored" subnet

Using the **Cloud Shell** terminal, ping the public address assigned to WEB1, replacing [PUBLIC_IP_WEB1] with the public IP address of "WEB1" which can be seen in the Cloud Console.

```
ping -c 4 [PUBLIC_IP_WEB1]
```

Packet Mirroring should duplicate and forward that traffic to the IDS VM and you should be able to see it in the packet capture of Step 1. Output on the IDS VM should be similar to this, where X.X.X.X is the source IP address of the ICMP requests. In the tcpdump output, you should expect to see the private IP of the WEBSERVER. Google Cloud makes the network translation at the edge.

```
00:55:32.980666 IP X.X.X.X > 172.21.0.2: ICMP echo request, id 3399, seq 0, length 64
00:55:32.980801 IP 172.21.0.2 > X.X.X.X: ICMP echo reply, id 3399, seq 0, length 64
00:55:33.968920 IP X.X.X.X > 172.21.0.2: ICMP echo request, id 3399, seq 1, length 64
00:55:33.968965 IP 172.21.0.2 > X.X.X.X: ICMP echo reply, id 3399, seq 1, length 64
```

```
00:55:34.980472 IP X.X.X.X > 172.21.0.2: ICMP echo request, id 3399, seq 2, length 64
00:55:34.980521 IP 172.21.0.2 > X.X.X.X: ICMP echo reply, id 3399, seq 2, length 64
00:55:35.986354 IP X.X.X.X > 172.21.0.2: ICMP echo request, id 3399, seq 3, length 64
00:55:35.986445 IP 172.21.0.2 > X.X.X.X: ICMP echo reply, id 3399, seq 3, length 64
```

The same should be true if you ping the public address of WEB2. Ping the public address assigned to WEB2, replacing [PUBLIC_IP_WEB2] with the public IP address of "WEB2".

```
ping -c 4 [PUBLIC_IP_WEB2]
```

PacketMirroring should forward that traffic to the IDS VM and you should be able to see it in the packet capture of Step 1. Output on the IDS VM should be similar to this. Please note that in the tcpdump output, you should expect to see the private IP of the WEBSERVER. Google Cloud makes the network translation at the edge.

```
00:00:45.309407 IP X.X.X.X > 172.21.0.3: ICMP echo request, id 25159, seq 0, length 64
00:00:45.309736 IP 172.21.0.3 > X.X.X.X: ICMP echo reply, id 25159, seq 0, length 64
00:00:46.309406 IP X.X.X.X > 172.21.0.3: ICMP echo request, id 25159, seq 1, length 64
00:00:46.309602 IP 172.21.0.3 > X.X.X.X: ICMP echo reply, id 25159, seq 1, length 64
00:00:47.306278 IP X.X.X.X > 172.21.0.3: ICMP echo request, id 25159, seq 2, length 64
00:00:47.306406 IP 172.21.0.3 > X.X.X.X: ICMP echo reply, id 25159, seq 2, length 64
00:00:48.314506 IP X.X.X.X > 172.21.0.3: ICMP echo request, id 25159, seq 3, length 64
00:00:48.314698 IP 172.21.0.3 > X.X.X.X: ICMP echo reply, id 25159, seq 3, length 64
```

To better demonstrate that Packet Mirroring shows more than just the Layer 3 headers, the following test will show a standard HTTP GET to one of the WEB servers, including the initial TCP 3-way handshake.

Open a new tab in your browser and open the public address assigned to WEB1 with the http protocol. The "curl" utility from Cloud Console may also be used, if you prefer.

Replace [PUBLIC_IP_WEB1] with the public IP address of "WEB1"

```
http://[PUBLIC_IP_WEB1]/
```

Packet Mirroring should forward that traffic to the IDS VM and you should be able to see it in the packet capture of Step 1.

Output on the IDS VM should be similar to this:

```
00:00:46.761748 IP X.X.X.60835 > 172.21.0.2.80: Flags [S]...
00:00:46.763037 IP 172.21.0.2.80 > X.X.X.60835: Flags [S.], ... ack ...
00:00:46.816407 IP X.X.X.60835 > 172.21.0.2.80: Flags [.], ack ...
00:00:46.823624 IP X.X.X.60835 > 172.21.0.2.80: Flags [P.], ... HTTP: GET / HTTP/1.1
00:00:46.823832 IP 172.21.0.2.80 > X.X.X.60835: Flags [.], ack ...
00:00:46.824549 IP 172.21.0.2.80 > X.X.X.60835: Flags [P.], ... HTTP: HTTP/1.1 200 OK
00:00:46.879214 IP X.X.X.60835 > 172.21.0.2.80: Flags [.], ack ...
00:00:46.888477 IP X.X.X.60835 > 172.21.0.2.80: Flags [F.], ...
00:00:46.888662 IP 172.21.0.2.80 > X.X.X.60835: Flags [F.], ... ack...
00:00:46.943915 IP X.X.X.60835 > 172.21.0.2.80: Flags [.], ack ...
```

The same should be true if you browse to the public address of WEB2. Replace [PUBLIC_IP_WEB2] with the public IP address of "WEB2"

```
http://[PUBLIC_IP_WEB2]/
```

Packet Mirroring should forward that traffic to the IDS VM and you should be able to see it in the packet capture of Step 1.

Output on the IDS VM should be similar to this.

```
00:00:58.819740 IP X.X.X.X.62335 > 172.21.0.3.80: Flags [S]...
00:00:58.820027 IP 172.21.0.3.80 > X.X.X.X.62335: Flags [S.], ... ack ...
00:00:58.875301 IP X.X.X.X.62335 > 172.21.0.3.80: Flags [.], ack ...
00:00:58.875329 IP X.X.X.X.62335 > 172.21.0.3.80: Flags [P.], ... HTTP: GET / HTTP/1.1
00:00:58.875448 IP 172.21.0.3.80 > X.X.X.X.62335: Flags [.], ack ...
00:00:58.876204 IP 172.21.0.3.80 > X.X.X.X.62335: Flags [P.], ... HTTP: HTTP/1.1 200 OK
00:00:58.929015 IP X.X.X.X.62335 > 172.21.0.3.80: Flags [.], ack ...
00:00:58.929047 IP X.X.X.X.62335 > 172.21.0.3.80: Flags [F.], ...
00:00:58.929244 IP 172.21.0.3.80 > X.X.X.X.62335: Flags [F.], ... ack...
00:00:58.993844 IP X.X.X.X.62335 > 172.21.0.3.80: Flags [.], ack ...
```

Type `ctrl+c` inside the IDS VM to exit the tcpdump.

Test Suricata IDS inspection and alerts

The final section of this lab is to test Packet Mirroring integration with the Open Source IDS Suricata. Take a minute to review the 4 Suricata rules set to alert from Step4 of the "Configure and Review Suricata" section:

```
####RULES#####
#UDP ALERTS
alert udp $HOME_NET any -> 8.8.8.8 53 (msg:"BAD UDP DNS REQUEST"; sid:99996; rev:1;)

#HTTP ALERTS
alert http any any -> $HOME_NET 80 (msg:"BAD HTTP PHP REQUEST"; http.uri;
content:"index.php"; sid:99997; rev:1;)

#ICMP ALERTS
alert icmp any any -> $HOME_NET any (msg:"BAD ICMP"; sid:99998; rev:1;)

#TCP ALERTS
alert tcp $HOME_NET any -> any 6667 (msg:"BAD TCP 6667 REQUEST"; sid:99999; rev:1;)
```

The following 4 steps will have you generate network traffic that trigger each of these rules. Alerts for each of them should be seen in the Suricata event log file.

Note: Make sure that you have opened up ssh windows for both the IDS VM and a WEB server VM. You will need to view BOTH simultaneously to complete this section.

TEST1 and TEST2 will be initiated from the WEBSERVER and test *egress* traffic.

TEST3 and TEST4 will be initiated Cloud Shell and test *ingress* traffic.

TEST 1: Test egress UDP rule/alert.

Run the following command from one of the **WEB servers** to generate egress DNS traffic:

```
dig @8.8.8.8 example.com
```

Now, view the alert in the Suricata event log file on the **IDS VM**.

Switch to the ssh window for the IDS VM

Run the following command in the SSH window for the IDS VM:

```
egrep "BAD UDP DNS" /var/log/suricata/eve.json
```

The log entry should be similar to this:

```
@GCP: {"timestamp":"2020-08-
14T01:23:14.903210+0000","flow_id":412864167987242,"in_iface":"ens4","event_type":"aler
t","src_ip":"172.21.0.2","src_port":47020,"dest_ip":"8.8.8.8","dest_port":53,"proto":"U
DP","alert":{"action":"allowed","gid":1,"signature_id":99996,"rev":1,"signature":"BAD
UDP DNS
REQUEST","category":"","severity":3},"dns":{"query":[{"type":"query","id":17268,"rrname
":"EXAMPLE.COM","rrtype":"A","tx_id":0}]},"app_proto":"dns","flow":{"pkts_toserver":1,"
pkts_toclient":0,"bytes_toserver":82,"bytes_toclient":0,"start":"2020-08-
19T18:23:14.903210+0000"}}
```

TEST 2: Test egress "TCP" rule/alert

Run the following command from one of the **WEB servers** to generate Egress TCP traffic:

```
telnet 100.64.1.1 6667
```

type `ctrl+c` to exit.

Now, view the alert in the Suricata event log file on the IDS VM.

Switch to the ssh window for the IDS VM

Run the following command from the SSH window of the IDS VM:

```
egrep "BAD TCP" /var/log/suricata/eve.json
```

The log entry should be similar to this:

```
@GCP: {"timestamp":"2020-08-
14T01:27:45.058526+0000","flow_id":479376049300638,"in_iface":"ens4","event_type":"aler
t","src_ip":"172.21.0.2","src_port":36168,"dest_ip":"100.64.1.1","dest_port":6667,"prot
o":"TCP","alert":{"action":"allowed","gid":1,"signature_id":99999,"rev":1,"signature":"
BAD TCP 6667
REQUEST","category":"","severity":3},"flow":{"pkts_toserver":1,"pkts_toclient":0,"bytes
_toserver":74,"bytes_toclient":0,"start":"2020-08-19T18:27:45.058526+0000"}}
```

TEST 3: Test ingress "ICMP" rule/alert

Run the following command from **Cloud Shell** to generate INGRESS ICMP traffic.

Replace [PUBLIC_IP_WEB1] with the public IP address of "WEB1".

```
ping -c 3 [PUBLIC_IP_WEB1]
```

Now, view the alert in the Suricata event log file on the **IDS VM**:

```
egrep "BAD ICMP" /var/log/suricata/eve.json
```

The log entry should be similar to this:

```
@GCP: {"timestamp":"2020-08-
14T01:24:46.065250+0000","flow_id":1114966772874978,"in_iface":"ens4","event_type":"ale
rt","src_ip":"X.X.X.X","dest_ip":"172.21.0.2","proto":"ICMP","icmp_type":8,"icmp_code":
0,"alert":{"action":"allowed","gid":1,"signature_id":99998,"rev":1,"signature":"BAD
ICMP","category":"","severity":3},"flow":{"pkts_toserver":1,"pkts_toclient":0,"bytes_to
server":98,"bytes_toclient":0,"start":"2020-08-19T18:24:46.065250+0000"}}
```

TEST 4: Test ingress "HTTP" rule/alert.

Using the web browser of your local workstation or curl in Cloud Shell, browse the public address assigned to WEB1 for the page "***index.php***", using the http protocol. Replace [PUBLIC_IP_WEB1] with the public IP address of "WEB1".

```
http://[PUBLIC_IP_WEB1]/index.php
```

Now, view the alert in the Suricata event log file on the **IDS VM**:

```
egrep "BAD HTTP" /var/log/suricata/eve.json
```

The log entry should be similar to this:

```
@GCP: {"timestamp":"2020-08-
14T01:26:36.794124+0000","flow_id":1901424684045611,"in_iface":"ens4","event_type":"ale
rt","src_ip":"X.X.X.X","src_port":61132,"dest_ip":"172.21.0.3","dest_port":80,"proto":"
TCP","tx_id":0,"alert":{"action":"allowed","gid":1,"signature_id":99997,"rev":1,"signat
ure":"BAD HTTP PHP
REQUEST","category":"","severity":3},"http":{"hostname":"PUBLIC_IP_WEB1","url":"\/index
.php","http_user_agent":"curl\/7.64.1","http_content_type":"text\/html","http_method":"
GET","protocol":"HTTP\/1.1","status":404,"length":275},"app_proto":"http","flow":{"pkts
_toserver":7,"pkts_toclient":6,"bytes_toserver":658,"bytes_toclient":1284,"start":"2020
-08-19T18:26:36.660779+0000"}}
```

Congratulations!

This completes the lab for Google Cloud Packet Mirroring use with the Open Source IDS Suricata.

Finish Your Quest

This self-paced lab is part of the Qwiklabs Networking in the Google Cloud and Security & Identity Fundamentals Quests. A Quest is a series of related labs that form a learning path. Completing a Quest earns you the badge above, to recognize your achievement. You can make your badge (or badges) public and link to them in your online resume or social media account. Enroll in a Quest and get immediate completion credit if you've taken this lab. See other available Qwiklabs Quests.

Take your next lab

Continue your Quest with Multiple VPC Networks, or check out these suggestions:

- Setting Up a Private Kubernetes Cluster
- Getting Started with Cloud KMS

Next Steps / Learn More

For additional information regarding Packet Mirroring, see:

*Packet Mirroring Overview: https://cloud.google.com/vpc/docs/packet-mirroring
*Using Packet Mirroring: https://cloud.google.com/vpc/docs/using-packet-mirroring
For more information about Suricata, see https://suricata-ids.org/

Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. Our classes include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. Certifications help you validate and prove your skill and expertise in Google Cloud technologies.
Manual Last Updated December 22, 2020
Lab Last Tested December 22, 2020