
Nubeva TLS Documentation

Release 1.2

Product

Jan 27, 2021

1	Nubeva TLS Overview	1
2	Getting Started	5
2.1	First Time Users	5
3	TLS Key Extraction	7
3.1	Installing Nubeva Sensors	7
3.2	Creating Source Groups	10
3.3	Launching Decryptors	12
4	Key Extraction QuickStart	15
5	Projects and Key DBs	17
5.1	Set up a Private Key DB	18
6	Set up a Fast Key DB	21
6.1	Sensor Container	21
6.2	Traffic Generator - Optional	22
6.3	Fast Key DB	22
7	TLS Key Record Formats	25
8	Modifying Project Elements	29
9	Packet Mirroring	31
9.1	Creating Destinations	31
9.2	Creating Connections	32
9.3	Decrypting AWS VPC Traffic Mirroring	34
9.4	Monitoring Sensors and Decryptors	35
10	TLS API	37
11	Deploying Security Tools	39
11.1	Tool Launcher	39
11.2	Moloch	41
11.3	ntopng	43
11.4	Suricata	45
11.5	Wireshark	47

11.6	Zeek	49
12	Help and Support	53
13	Frequently Asked Questions	55
13.1	OS Vendor Changes	55
13.2	Government/Compliance Issues	56
13.3	Interaction with AV/Malware/EDR Products	56
13.4	TPM-type solutions	56
13.5	Protocol Updates, Changes & Additions	56
14	Installing Docker	57
14.1	Ubuntu 18.04	57
14.2	AWS Linux	57
14.3	AWS Linux 2	58
14.4	Cent OS 7	58
14.5	Red Hat Enterprise Linux 7.5 (RHEL)	58
14.6	Cloud Provider Instructions	58
15	Ciphers Supported	59
16	Supported Libraries and Operating Systems	61
16.1	OpenSSL	61
16.2	NSS Libraries	65
16.3	WolfSSL	67
16.4	Linux	68
16.5	MS Windows	68
17	Berkeley Packet Filters	71
18	AWS	79
18.1	Using AWS VPC Traffic Mirrors	79
18.2	Creating AWS IAM Role for Custom Tag Support	79

Nubeva TLS Overview

Welcome to Nubeva's TLS, a complete cloud traffic visibility solution for AWS, Azure and GCP.

Traffic visibility is a crucial component in securing the business and keeping systems operational. However, network monitoring has been blinded in the cloud. Not only is the infrastructure used for monitoring unaccessible in the cloud, more than 75% of cloud traffic is encrypted. Traditional encryption services cannot adapt and cannot support the ephemeral nature of cloud workloads, and the newer encryption standards which enforce perfect forward secrecy and preclude 'man in the middle' encryption techniques. IT teams are no longer able to acquire, process and distribute decrypted packet-level cloud traffic to their selected tools. Consequently, the move to the cloud creates significant blind-spots and loss of ROI on vital tools that are powerless without access to packet-level cloud data.

Nubeva's TLS Visibility Solution is a Software as a Service (SaaS) offering that provides complete packet visibility into any public cloud with breakthrough TLS decryption capabilities that have been designed specifically for the cloud. Nubeva TLS Visibility Solution's architecture is comprised of three building blocks: a Nubeva Manager (console), Nubeva Sensors (or Sensors) and Nubeva Decryptors (or Decryptors). The architecture is scalable, secure, and traffic visibility is achieved without sending decrypted packets across the network. Nubeva Sensors discover TLS/SSL session keys and forward them to secure storage. Sensors also mirror packets within a cloud instance and forwards them to Nubeva Decryptors running on security and analysis tool instances. Nubeva TLS Decrypt works with any cloud packet broker solution including AWS VPC Traffic Mirroring. Nubeva Decryptors retrieve session keys from the secure storage, based on the session identifiers in the packet flows they receive, and produce both encrypted and decrypted traffic flows on an interface which a security tool running on the same cloud instance can access.

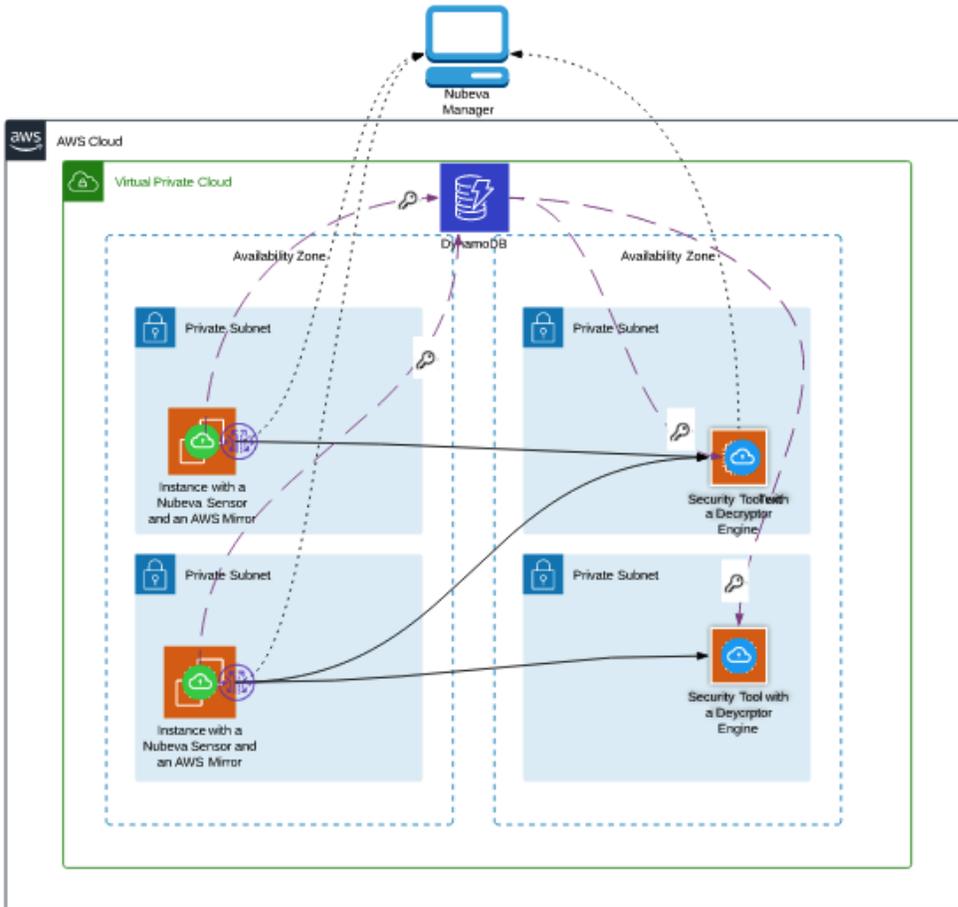


Figure 1: TLS Visibility Solution architecture with AWS VPC Traffic Mirroring

Figure 1 depicts a sample deployment in an AWS cloud using AWS VPC Traffic Mirroring. Dotted lines represent control messages, dashed line represent session keys, and solid lines represent mirrored traffic.

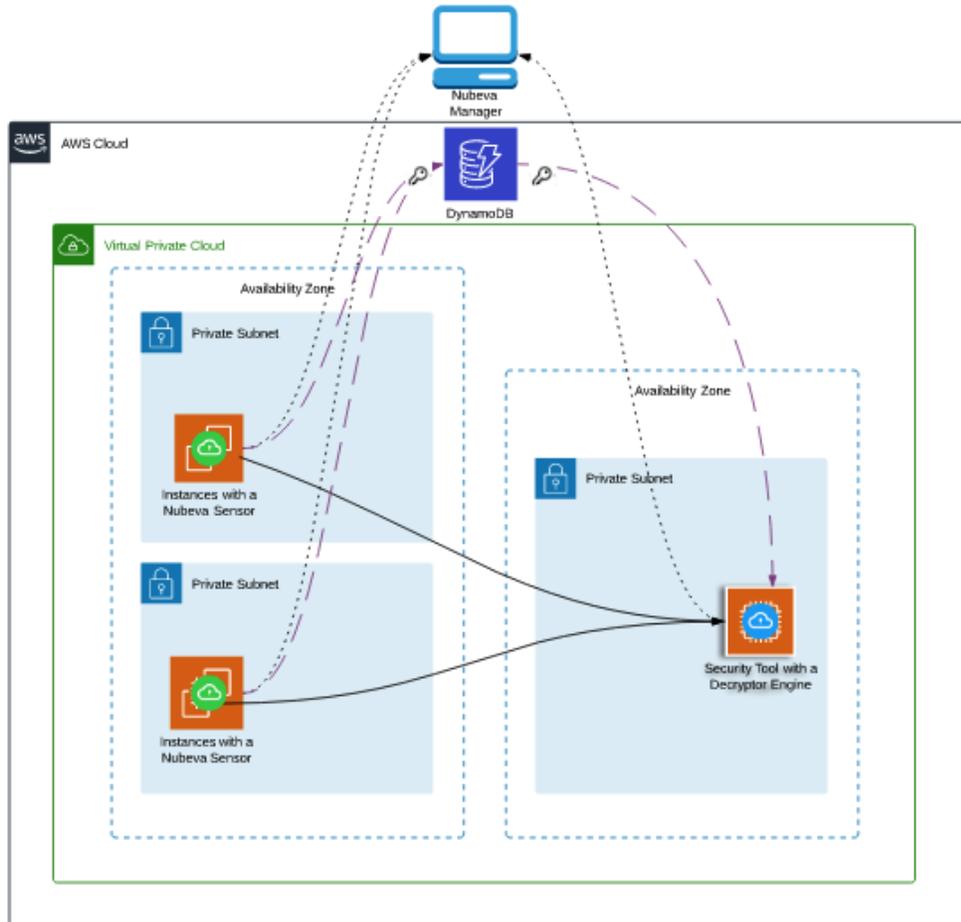


Figure 2: TLS Visibility services architecture with Nubeva Sensors discovering keys and mirroring traffic.

Note: Nubeva Decryptors handle the synchronization of keys with packet flows, assuring that all the traffic received is matched with keys, and is fully decrypted.

When any instance containing a Nubeva Sensor or a Nubeva Decryptor launches, the sensor/decryptor will automatically connect to the Nubeva Manager (console) and register itself, obtain configuration updates and automatically install software updates when upgrades are available. Sensors and Decryptors use HTTPS to make REST API calls to the Cloud Console. Control traffic always originates at sensors and decryptors. Data plane traffic (mirrored traffic) is routed based on the users' network configurations. Mirrored packets are never sent to the Cloud Console. The control plane does not directly modify, nor does it require the user to modify networks or security setting, save for allowing outbound HTTPS (TCP port 443) from subnets containing sensors or decryptors.

The following URLs and IP addresses should be accessible for the sensors/decryptors to connect:

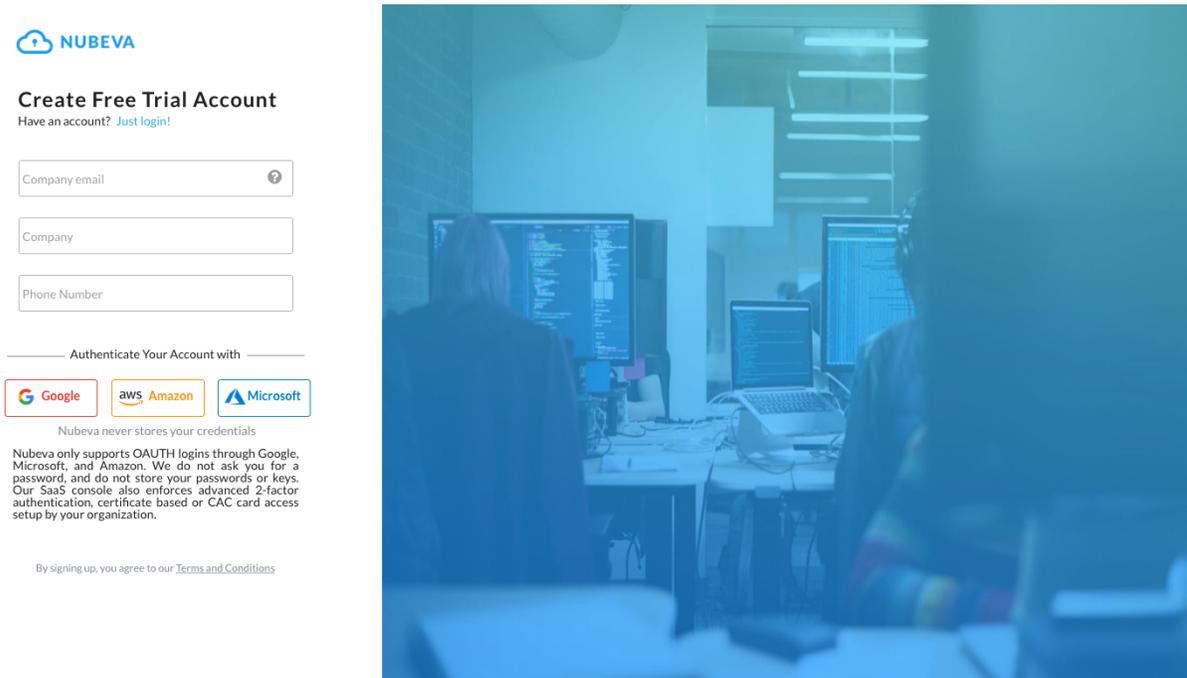
```
https://i.nuos.io/api/1.1/wf
https://rvs.nuos.io
13.248.140.181
52.183.93.152
```

Note: To set up AWS VPC traffic mirroring sessions please review [. Additional information is available on the \[.\]\(#\)](#)

2.1 First Time Users

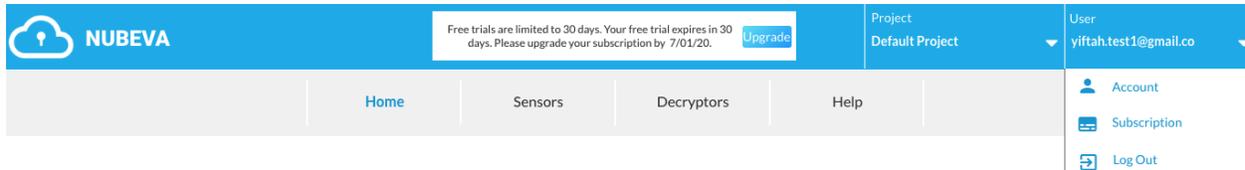
To get started with Nubeva TLS Visibility Solution, start by creating an account and log in on the *Nubeva Manager (console)*:

1. Navigate to and select 'Login' from the main menu.
2. First-time users will be prompted to create an account. Go ahead and use one of the OAuth partners to log in.



Note: Nubeva only supports OAUTH logins through Google, Microsoft, and Amazon. We do not ask you for a password, and do not store your passwords or keys.

- When you log in to Nubeva Manager for the first time, you will see a helper block which provides useful links to on-line guides and videos. You can revisit this information on the Help page. You are automatically subscribed to a free trial for 30 days. You may upgrade your subscription at any time before the end of the trial by clicking the `upgrade` button in the main menu bar, or by selecting the `Subscription` option from the account menu as shown in the figure below:



When you log in a 'Default Project' is created automatically for you. As part of creating a project, Nubeva Manager also creates a DynamoDB table to store session keys which the Nubeva Sensors extract.

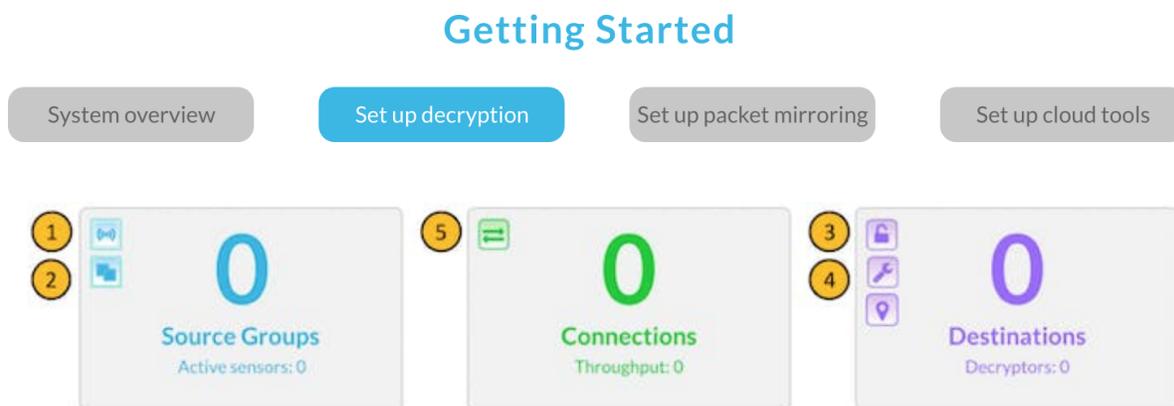
Tip: You should replaced the default DynamoDB table with one in your own account when going to production. Instructions are provided later in this section.

TLS Key Extraction

Extracting keys and decrypting workloads requires five steps:

1. Installing Nubeva Sensors
2. Creating Source Groups
3. Installing Decryptor Engines
4. Configuring Cloud Tools
5. Mirroring traffic

The visual tiggers for each step are marked in the figure below:



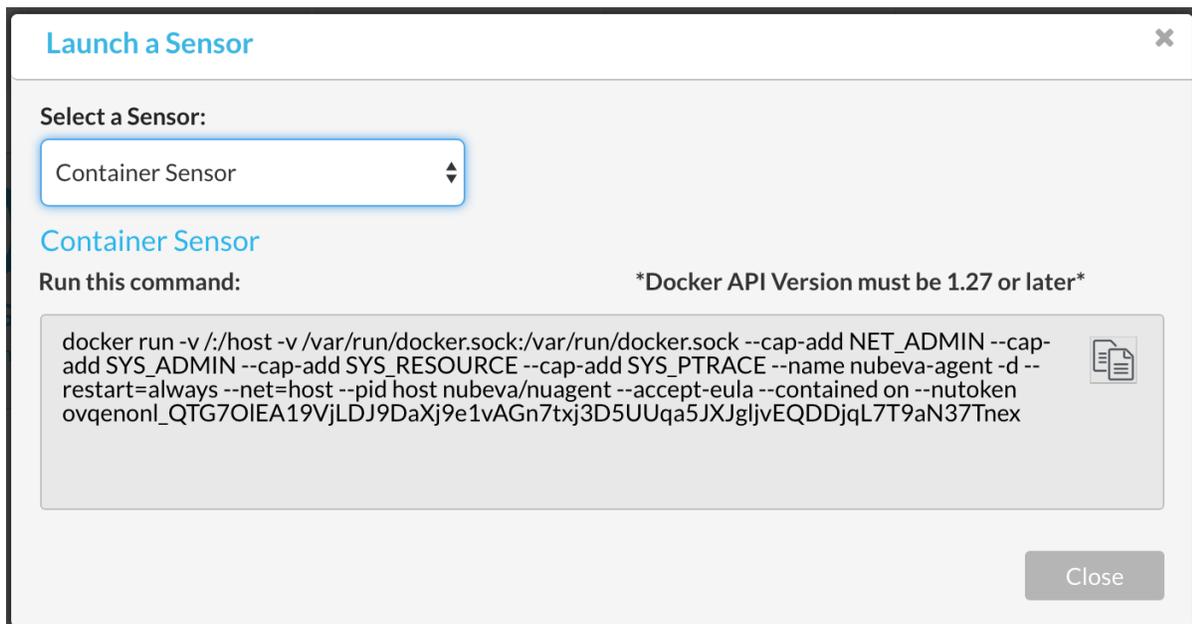
3.1 Installing Nubeva Sensors

As shown in figure 1 in the previous section Nubeva Sensors mirror traffic as well as extract session keys from encrypted traffic. Nubeva Sensors are available for *Linux* and *MS Windows*.

1. Linux sensors are available as Docker containers as well as a standalone installation. You can skip this step if your instance is already running the required version of Docker. If you need to install docker please refer to our instructions.



2. Click on the top icon in the left corner of the Source Group box.
3. This will pop up a box similar to the figure below. Select 'Container Sensor' from the drop-down menu. Click the button on the right to copy the installation command.



4. Paste this command into a command shell on the cloud instance. The sensor container will automatically download and install.
5. About 10-20 seconds after installation, the `Active sensors` counter in the `Source Groups` box will increase by 1 indicating that the sensor is active.
7. To launch a native Linux installer select the options shown in the figure below:



7. To launch a Windows sensor, select “Windows Sensor” from the drop-down.



8. Paste this command into a command shell on the cloud instance. The sensor installer will automatically download and install from Docker Hub.

Note: The last command is:

```
& "$DownloadDir\installer.exe" NUTOKEN_USERINPUT=$NubevaTok API_URL_ARG=$
↵{InstallerArg} /q;
```

To enable detailed logging replace the last command with:

```
& "$DownloadDir\installer.exe" NUTOKEN_USERINPUT=$NubevaTok API_URL_ARG=$
↔{InstallerArg} /q /L*v "$DownloadDir\installer.log";
"Debugging logs are in: '$DownloadDir\installer.log'
```

To uninstall a Windows sensor run:

```
wmic product where "description='Nubeva Sensor' " uninstall
```

Tip:

Nubeva sensors can be run as daemonsets in Kubernetes clusters. Nubeva sensors require Kubernetes versions 1.11 or greater. If you would like to extract keys from nodes running in a Kubernetes cluster please do the following:

1. Download <https://nubevalabs.s3.amazonaws.com/nuAgentDaemonSet.yaml>
2. Edit the file and replace the value of the `nutoken` placeholder with the value that is used in your project. This is the value of the `nutoken` parameter in the sensor-launch dialogs depicted above.
3. Run

```
kubectl apply -f nuAgentDaemonSet.yaml
```

The next step is to configure source groups.

3.2 Creating Source Groups

Source groups instruct sensors to extract keys. Sensors are grouped based on their *metadata* and *custom tags*. A sensor must be included in a source group in order to extract keys. The following links provide additional information: , , .

As new sensors with matching meta-data and/or custom tags appear, they are automatically added to the source group. This is a powerful adaptation to the elastic nature of cloud environments. For instance, if you routinely spin up/down instances with web scaling events, selecting filters such as AMI type, VPC, subnet, or custom tag values, will ensure that any new sensors that appear will be immediately added to the source groups and start extracting keys.

Note: For additional security and scaling, Nubeva supports Amazon DynamoDB as a Private KeyDB. Sensors store session keys in a private DynamoDB table in your account. Other cloud DBs such as Azure Cosmos DB or Google Cloud Data-store are on the road-map. Please contact us if you are interested in those platforms.

: All communication of keys and associated meta-data is encrypted both in transit and at rest. The session keys for the sensor to KeyDB communication is NEVER saved. All user data stored in Amazon DynamoDB is fully encrypted at rest. DynamoDB encryption at rest provides enhanced security by encrypting all data at rest using encryption keys stored in AWS Key Management Service (AWS KMS). Nubeva has no access to this Private KeyDB running on Amazon DynamoDB.

: For additional security Nubeva suggests that you use VPC endpoints for all connectivity to a Private KeyDB running on Amazon DynamoDB.. When you create a VPC endpoint for DynamoDB, any requests to a DynamoDB endpoint within the Region (for example, `dynamodb.us-west-2.amazonaws.com`) are routed to a private DynamoDB endpoint within the Amazon network. You don't need to modify your applications running on EC2 instances in your VPC. The endpoint name remains the same, but the route to DynamoDB stays entirely within the Amazon network, and does not access the public Internet.

These are the steps to create a source group:



1. Click on the lower icon at the top right of the “Source Group” box. This will load the Source Group editing window.

Add Source Group ✕

Properties

Source Group Name *

Description

Enable TLS Key Extraction

Key extraction is enabled using Nubeva's hosted KeyDepot

Source Inclusion Policy

Filter Type

Metadata Category

Equals

Value

+

Save

Available Sources

Instance ID	Cloud	Cloud Region	IPV4	Private Hostname	Uptime (hrs)
i-0b64b6fdf40520df6	AWS	us-east-1	172.31.52.75	ip-172-31-52-75.ec2.internal	26.2

2. Name the new source group.
3. Click on the Filter Type (leftmost) drop down to select either ‘Metadata’ or ‘Custom Tags’.

Note: On AWS permission has to be given to describe all instances because the scope of DescribeInstances cannot be limited to a single instance. *Creating an AWS IAM role for custom tag support*

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances"
      ],
      "Resource": "*"
    }
  ]
}

```

4. Click on the Metadata Category in the “Source Inclusion Policy” and choose something. Select the condition and select the values. Then click the ‘+’ button.

5. You can add multiple conditions.
6. Select save. This will return you to the dashboard.

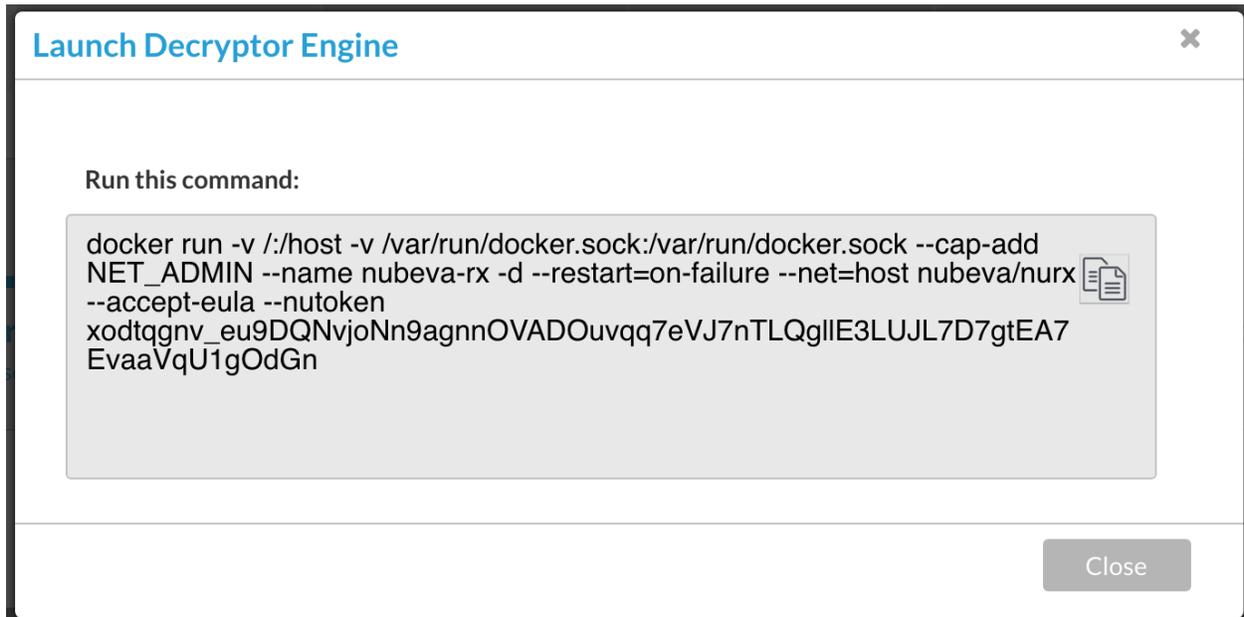
Tip: By default key extraction is turned on. The check box in the upper right corner allows you to turn key discovery on/off for the sensors included in the source group.

3.3 Launching Decryptors

1. Docker is a prerequisite to running containers. You can skip this step If your instance is already running the required version of Docker. If you need to install Docker please refer to for instructions.



2. Click on the “Lock” (top) icon on the top left of the Destination Group box.
3. This will display the popup depicted in figure below. Click the button on the right to copy the installation command.



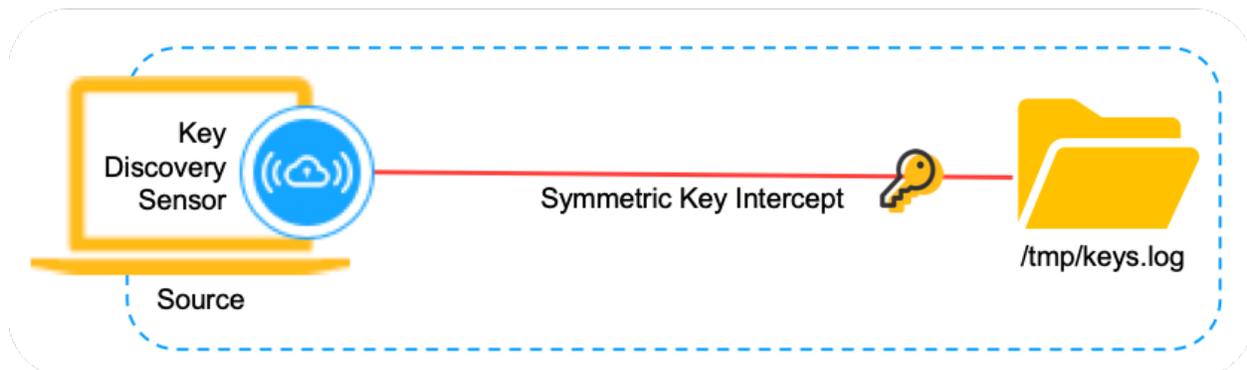
4. Paste this command into a command shell on the decryptor cloud instance. The decryptor container will automatically download and install.
5. About 10-20 seconds after installation, the decryptor counter in the Destination Groups box will increase by 1 indicating that the decryptor is active.
6. If you would like to launch a cloud security tool to process the decrypted packets, please refer to the [Tool Launcher](#) section.

Note: Refer to *Deploying Security Tools* for comprehensive instructions for launching security tools based on the .

The final step is to mirror traffic to your decryptor. This can be done by using AWS VPC Traffic Mirrors or by defining destinations and connections between source groups and destinations.

Key Extraction QuickStart

To familiarize yourself with basic key extraction you can run a `Simple Sensor` configuration. This configuration runs a single sensor instance extracting keys to a local file on the same instance where the sensor is running.



You can launch a simple sensor configuration by adding a paramters to a Container Sensor's launch command. A full command example is listed below.

```
docker run -v /:/host -v /var/run/docker.sock:/var/run/docker.sock --cap-add NET_
↪ADMIN --cap-add SYS_ADMIN --cap-add SYS_RESOURCE --cap-add SYS_PTRACE --name nubeva-
↪agent -d --restart=always --net=host --pid host nubeva/nuagent --accept-eula --
↪contained on --nutoken <Nubeva-Token> --sslcredobj_
↪eyJ0eXB1IjoibGNsIiwicmVnaW9uIjoiL2hvc3QvdG1wLyIsImRvbWVpbiI6ImtleXMubG9nIn0K
```

The `Nubeva-Token` will be included in the command when you copy from the sensor-launch dialog (*Installing Nubeva Sensors*). The last parameter which is added to the launch command is listed below. This parameter instructs the sensor to store keys in `/tmp/keys.log`.

```
--sslcredobj_
↪eyJ0eXB1IjoibGNsIiwicmVnaW9uIjoiL2hvc3QvdG1wLyIsImRvbWVpbiI6ImtleXMubG9nIn0K
```

You can install `nubevalab/tlsgenerator` which is a standard traffic generator container. The command to launch the generator is:

```
docker run -dti nubevalab/tlsgenerator
```

In addition you may install a dockerized version of Wireshark that is accessible using HTTPS. To deploy the container run:

```
docker run -v /tmp:/keys -p 14500:14500 --restart unless-stopped -dti --cap-add NET_
↳ADMIN --net=host --name wireshark ffeldhaus/wireshark
```

The directory of the keys file is mapped to the /keys directory on the Wireshark container, so the key file will be /keys/keys.log.

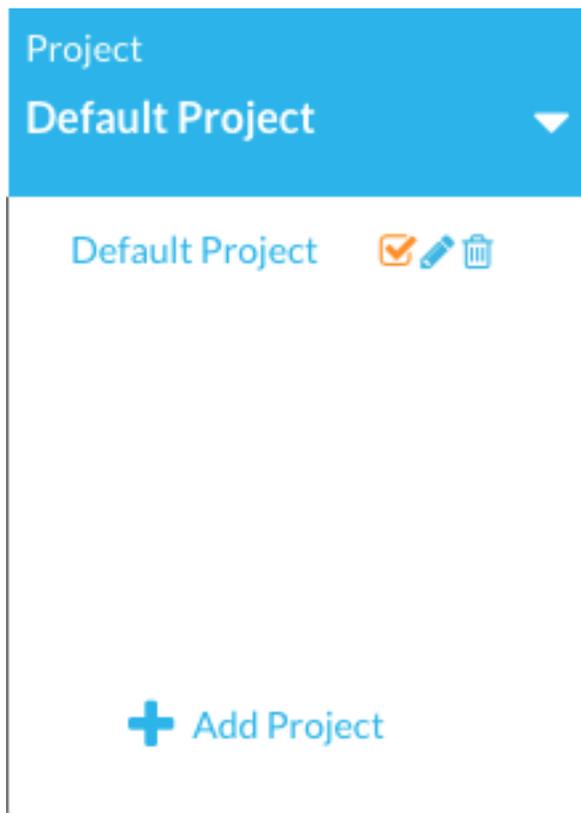
Note: The user-name and password for accessing Wireshark are `wireshark`

You can deploy all three elements using a cloud formation template stored at [.](#) To launch the template you need a VPC and a Nubeva Token. The CloudFormation template installs all three containers and sets up a cron job to run a generator container every minute. The figure below depicts the containers running.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d4df1946467c	nubevalab/tlsgenerator	"/opt/nubevaTools/tl..."	2 seconds ago	Up 1 second		distracted_moser
3ebd1989aa74	nubevalab/tlsgenerator	"/opt/nubevaTools/tl..."	About a minute ago	Up About a minute		infallible_hamilton
883b38e8f0e7	ffeldhaus/wireshark	"/docker-entrypoint..."	About an hour ago	Up About an hour		wireshark
4c3e7ffa7872	nubeva/nuagent	"/app/nuagent --acce..."	About an hour ago	Up 3 minutes		nubeva-agent

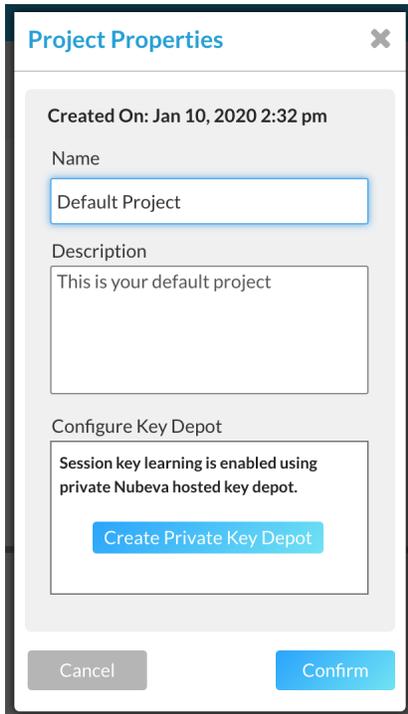
Projects and Key DBs

Projects are groupings of key extraction work-flows. A sensor can belong to only one project. Projects allow you to easily manage deployments as they grow. You can add/remove/select projects from the 'Project' option in the main menu.



5.1 Set up a Private Key DB

A private key DB is automatically configured for each of your projects in a DynamoDB table hosted by Nubeva.



Tip: Nubeva TLS supports AWS, Azure and GCP. The key database is a DynamoDB table in AWS.

The project properties dialog indicates which DynamoDB table is being used. The figure shows that the table is in Nubeva's account. To replace the default table with a table in your own account, click the `Create Private Key DB` button.

This step is not required for POC but recommended when you go to production.

Select the region and click `Launch DB`. This will launch a Cloud Formation template. The template creates IAM resources for writing and reading from the Key DB. Acknowledge that you allow these roles to be created:

When the template is done select the `Outputs` tab and click the URL in the field `SendtoNubeva`:

This operation navigates back to the home page of your current project and registers the encrypted credentials.

Note: If you delete the private Key DB, you can always create another. If you delete a private Key DB and do not create another, then sensors will no longer extract session keys.

Private Key Depot Setup ✕

This will set up a DynamoDB table for this project in your account.

Select a Cloud aws

Select a Region us-east-1 (N. Virginia)

Clicking 'Launch DB' runs a cloud formation template in a new browser tab. When the template is done, please click 'Outputs' and then click the "SendToNubeva" URL ([Documentation](#))

NubevaCreds

Stack info | Events | Resources | **Outputs** | Parameters | Template

1. Click Outputs

Delete Update Stack actions ▼

Outputs (1) ↻

🔍 Search outputs

Key	Value
SendToNubeva	2. Then click the URL https://l.nuos.io/topo?ProjectID=1559673686x701667641017502302461626&ssldbreadauth=eyJ0eXBlljoiZGRllwIZG9tYWluljoiTnV

Launch DB

Close

Capabilities

ⓘ The following resource(s) require capabilities: [AWS::IAM::User, AWS::IAM::Role]

This template contains Identity and Access Management (IAM) resources. Check that you want to create each of these resources and that they have the minimum required permissions. In addition, they have custom names. Check that the custom names are unique within your AWS account. [Learn more.](#)

I acknowledge that AWS CloudFormation might create IAM resources with custom names.

Cancel

Create change set

Create stack

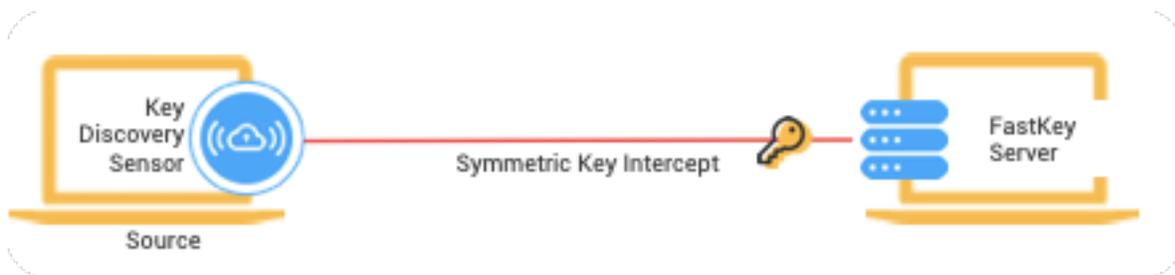
The screenshot shows the AWS CloudFormation console interface. On the left is a navigation sidebar with options like 'Stacks', 'Stack details', 'Change sets', 'Drifts', 'StackSets', 'Exports', 'Designer', 'Previous console', and 'Feedback'. The main area displays the 'NubevaCreds' stack details. The 'Outputs' tab is selected and highlighted with a red box. Above the 'Outputs' tab, there are buttons for 'Delete', 'Update', and 'Stack actions'. The 'Outputs (1)' section contains a search bar and a table with one output:

Key	Value
SendToNubeva	https://l.nuos.io/topo?ProjectID=1559673686x701667641017502302461626&ssldbreadauth=eyJ0eXBlljoizGRllwiZG9tYWluljoTrnV

Red annotations on the screenshot include: '1. Click Outputs' pointing to the 'Outputs' tab, and '2. Then click the URL' pointing to the URL value in the output table.

Set up a Fast Key DB

It is possible to launch sensors and decryptors with a Fast Key DB configuration. The configuration is depicted in the figure below. The Fast Key DB service runs on an instance you designate.



Sensors configured to use Fast Key DB send keys using DTLS to a defined destination. Keys are sent within 200 microseconds of generation. Fast Key DB includes two required components, sensor and Key-Depot.

If you are deploying via the CloudFormation template, a VPC and a Nubeva Token are required. 2 EC2 instances are used in the build. The Source has 2 docker containers and the Destination/Key DB has 1 container.

The easiest way to deploy Simple DB is with a CloudFormation template stored at <https://nubevalabs.s3.amazonaws.com/nubeva-fastkeydb.template.yaml>. The source can be found at <https://github.com/nubevalabs/templates>.

To launch the template you need a VPC and a Nubeva Token. The CloudFormation template runs two EC2 instances. The source instance runs a sensor and traffic generator. The destination instance runs a Fast Key DB Container.

You can launch a subset of the containers yourself. You will need at least a sensor container on the source, and a Simple Key DB on the destination. The following sections describe how to launch each of the containers.

6.1 Sensor Container

A sensor container is required on the source instance. This is a standard sensor that uses a special parameter specifying that it should **only** write keys to `key.nubedge.com` which must resolve (usually with `/etc/hosts`) to the destination instance.

```
docker run -v /:/host -v /var/run/docker.sock:/var/run/docker.sock --add-host key.
↪nubedge.com:<<KeyDB IP address>> --cap-add NET_ADMIN --cap-add SYS_ADMIN --cap-add
↪SYS_RESOURCE --cap-add SYS_PTRACE --name nubeva-agent -d --restart=always --
↪net=host --pid host nubeva/nuagent --accept-eula --contained on --nutoken <<your_
↪token>> --sslcredobj_
↪eyJ0eXB1IjoiZHRscyIsImRvbWFpbiI6ImtleS5udWJlZGdlLmNvbTo0NDMzIiwicmVnaW9uIjoidGVzdCIsmFrIjoidXNlci
```

Note that you need to specify the IP address of `key.nubedge.com`. You will also need to modify your `/etc/hosts` file.

The last parameter in the command instructs the sensor to send its keys to a Fast Key DB.

```
--sslcredobj_
↪eyJ0eXB1IjoiZHRscyIsImRvbWFpbiI6ImtleS5udWJlZGdlLmNvbTo0NDMzIiwicmVnaW9uIjoidGVzdCIsmFrIjoidXNlci
```

This string is a base64 encoded json object that can be generated using the following command:

```
echo '{"type":"dtls","domain":"key.nubedge.com:4433","region":"test","ak":"user","sk":
↪"password"}' | base64
```

Type “dtls” value means a Fast Key DB, ‘domain’ is the destination host name, ‘region’ is ‘test’, user, and password fields can be left the same. They are not used but they are required. You can change the domain to anything however you MUST have a valid cert. This docker container contains a valid cert for `key.nubedge.com`.

To run a Windows sensor that sends keys to `key.nubedge.com` run the following PowerShell command (make sure to replace the value of the Nubeva token with your own):

```
$DownloadDir = $env:TEMP; $BaseUrl="https://i.nuos.io/";
$InstallerArg="-baseurl ${BaseUrl}api/1.1/wf/ -sslcredobj_
↪eyJ0eXB1IjoiZHRscyIsImRvbWFpbiI6ImtleS5udWJlZGdlLmNvbTo0NDMzIiwicmVnaW9uIjoidGVzdCIsmFrIjoidXNlci
↪";
$NubevaTok="<Your Nubeva Token>";
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
Invoke-WebRequest -Uri "${BaseUrl}NubevaSensor.latest.setup.exe" -OutFile "
↪$DownloadDir\installer.exe";
& "$DownloadDir\installer.exe" NUTOKEN_USERINPUT=$NubevaTok API_URL_ARG=$
↪${InstallerArg} /q;
```

6.2 Traffic Generator - Optional

You can add a traffic generator container to the source, however this is not required. This is Nubeva’s standard traffic generator container. A container runs for approximately 60-120 seconds. You may use a cron job to run a container every minute. The actual docker command to run this generator once is:

```
docker run -dti nubevalab/tlsgenerator
```

6.3 Fast Key DB

`fastkeydb.py` is a python script that uses Flask to simulate Nubeva’s Fast Key DB REST API. The script uses DTLS as a method to receive keys quickly. Source is located in same templates repo noted above.

6.3.1 Linux Fast Key DB

You can deploy a Fast Key DB container with the following command:

```
docker run -p 4433:4433/TCP -p 4433:4433/UDP -dti --name fastkeydb nubevalab/fastkeydb
```

You can see keys by running:

```
curl https://key.nubedge.com:4433/dumpkeys
```

6.3.2 Windows Fast Key DB

Download `fastkeydb.py`. You will need to receive the necessary cert files `nubedge.ca`, `nubedge.key` and `nubedge.pem` from Nubeva. The Python script requires the `--certs-path` parameter to specify the directory path for the cert files e.g. `C:\Nubeva\`. Note that the last backslash is required.

Edit `C:\Windows\System32\drivers\etc\hosts` file and add an entry for `key.nubedge.com`:

```
127.0.0.1 key.nubedge.com
```

Install necessary Python libraries:

```
pip install flask
pip install dtls
```

Or

```
python -m pip install flask
python -m pip install dtls
```

Run the script

```
python fastkeydb.py --nssfile C:\<path to key log>\keys.log --certs-path C:\<path to ↵
↵certs>\
```

You can check that Fast Key DB is receiving keys accessing the URL below from a browser:

```
https://key.nubedge.com:4433/dumpkeys
```

TLS Key Record Formats

Sensors use a REST API to send keys to key depots. The rest API uses a JSON object. The object is a key:value pair, where the key is a ‘client random’ and the value is a set of key fields and a meta-data structure. Meta-data data is not required for decryption.

The structure is the same for TLS 1.2 and TLS 1.3 keys. Fields are populated based on the TLS protocol type. The names of the fields match the specification. The JSON object uses the following abbreviations:

- **MS:** master key - this field is populated when the type is “TLS 1.2”
- **CETS:** client early traffic secret - TLS 1.3
- **CHTS:** client handshake traffic secret - TLS 1.3
- **SHTS:** server handshake traffic secret - TLS 1.3
- **CTS0:** client traffic secret 0 - TLS 1.3
- **STS0:** server traffic secret 0 - TLS 1.3
- **XS:** exported secret - TLS 1.3

The figure below depicts a TLS 1.2 key object

```
{
  "07ad5a5d9745c73e599e2147c7bb471249ed427a209dae6b77a807e898b9e5ec": {
    "CETS": "",
    "CHTS": "",
    "CR": "07ad5a5d9745c73e599e2147c7bb471249ed427a209dae6b77a807e898b9e5ec",
    "CTS0": "",
    "LastUsed": "2020-06-10T23:59:58",
    "MD": {
      "command": "curl",
      "hostname": "ip-172-31-5-150.ec2.internal",
      "instance": "i-05f1e99c1cd9e3b28",
      "lib": "ossl",
      "pid": 22909
    }
  },
}
```

(continues on next page)

(continued from previous page)

```

    "MK":
    ↪ "8b48f8a3f80ee7bbf26166d20913ce5ff068924d23b79199adc1babc4385c084c9cccc143ffa19963db60e010157fe33
    ↪ ",
    "SHTS": "",
    "STS0": "",
    "Type": "1.2",
    "XS": ""
  }
}

```

The following figure depicts a TLS 1.3 key object

```

{
  "01fc0baa6eca082096d69f047e232ed762ba317b1e7392178ca8c2579c73c464": {
    "CETS": "",
    "CHTS":
    ↪ "34110bb15d8fad38f0e2cda16cebe43e5f30cf60066ab04bf6cabf9553b175c6e2d1a3005b1d1c5527d9d1ad9a750679
    ↪ ",
    "CR": "01fc0baa6eca082096d69f047e232ed762ba317b1e7392178ca8c2579c73c464",
    "CTS0":
    ↪ "1bfa4c5d2351a746fb4472f5ca0276c81864ac613f994b06570431c1751f3aa88a4a67a66375a403dc9792e913b830b8
    ↪ ",
    "LastUsed": "2020-06-11T00:01:43",
    "MD": { "command": "curl",
            "hostname": "ip-172-31-5-150.ec2.internal",
            "instance": "i-05f1e99c1cd9e3b28",
            "lib": "ossl",
            "pid": 23311},
    "MK": "",
    "SHTS":
    ↪ "a86ae64b0fe1bd9e065125768251fd279089d0e544e2200f2d3df87edc2692d45befce649fac006d0fab153f2365a41a
    ↪ ",
    "STS0":
    ↪ "4c84bdae94562490dc2371cf32a48489fe03e89f7c464efae93afdc1df522d774b875ffabf95ce8e35cdf9b89773855a
    ↪ ",
    "Type": "1.3",
    "XS":
    ↪ "7dce5eec25e6e48a4f0a6bad9ece783fe7ef208d2508a1112b9a074d000e8cc9425ff2d59ac99b33715c2bdf98547510
    ↪ "
  }
}

```

Two communication channels are used when sensors send keys to a DTLS key target. A control channel uses a REST API to send keys. A low latency protocol uses DTLS to send a binary representation of key records. The structure of the binary format is the following:

- **Byte 0:** Version: a running counter of Nubeva’s low latency protocol version. The current version can be 0x1. This is the same number returned with the bearer token.
- **Byte 1:** Type indicates what function is requested. 0xC8 (200) = putkey
- **Bytes 2-3:** Protocol Version: 0303 = TLS 1.2, 0304 = TLS 1.3
- **Bytes 4-5:** Length: indicates the length of the keys + the length of the bearer token (64 + 32 + 48 + 384 = 528).
- **Bytes 6-69:** Bearer-token (64 bytes)
- **Bytes 70-101:** Client random (32 bytes)
- **Bytes 102-149:** Master key (48 bytes). 0’s if ‘protocol version’ is 0304

- **Bytes 150-213:** CETS (client early traffic secret) 0's if 'protocol version' is 0303
- **Bytes 214-277:** CHTS (client handshake traffic secret) 0's if 'protocol version' is 0303
- **Bytes 278-341:** SHTS (server handshake traffic secret) 0's if 'protocol version' is 0303
- **Bytes 342-405:** CTSZ (client traffic secret 0) 0's if 'protocol version' is 0303
- **Bytes 406-469:** STSZ (server traffic secret 0) 0's if 'protocol version' is 0303
- **Bytes 470-533:** XS (exported secret) 0's if 'protocol version' is 0303

Sample code that shows how to receive REST API and UDP messages for storing keys is available at <https://nubevalabs.s3.amazonaws.com/dtlskeydb/dtlskeydb.py>. Instructions to run the script are provided in the DTLS Key DB section at the bottom of the previous page.

Note: Since the client random value is used as the unique key for looking up session master keys, it is possible to store the same master key more than once. This assures that TLS session renewals are supported as well.

Modifying Project Elements

You can modify or delete any element by clicking an element and selecting an option from the pop-up menu:

1. Source Groups:



View/Modify Source Groups

Select `Group Properties` to view and modify a source group, `Delete Group` to remove a source group and its connections.

2. Destinations:



3. Connections:



View/Modify Connections

Select `Connection Properties` to view and modify a connection, `Delete Connection` to remove it.

Packet Mirroring

Note: You can set up traffic mirroring without running Nubeva Decryptors. The traffic reaching the target security tool will be encrypted. If you are using AWS VPC Traffic Mirrors you can skip to *Decrypting AWS VPC Traffic Mirroring*.

9.1 Creating Destinations

Destinations are define the IP addresses of instances running Cloud Tools. There are many packet inspection and processing tools in the open source community as well as many vendor offerings. Before we create a destination in the UI, we need to have an actual tool.

These are the steps to create a destination that points to a tool:



1. Click on the “Marker” (bottom) icon on the “Destinations” box.
2. The next screen allows you to define the properties of this new destination group.

3. Name the destination.
4. Choose between a single tool or a set of tools that you want to load share against. For a load share environment, insert multiple comma-separated IP addresses into the field and the traffic will be load-shared among the defined tools.

Note: If you need more advanced load-balancing, you can use the front-end IP address of a cloud load balancer as the IP and configure this LB via the cloud portals. To enable VXLAN traffic to your destination inbound traffic should be enabled on UDP port 4789.

Custom UDP Rule	UDP	4789	0.0.0.0/0
Custom UDP Rule	UDP	4789	::/0

One of the simplest tools you can use is `tcpdump`. This is a simple Unix command that takes all data received on an interface and displays it on the screen. You also use it to write this traffic to a file.

Create a Unix instance in your cloud provider. Connect to it and issue the command:

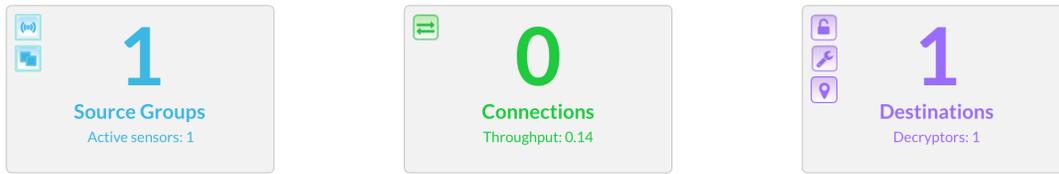
```
tcpdump -na -i eth0 port 4789
```

This will start a `tcpdump` session which will display all mirrored traffic on your default interface but it will not show your SSH session traffic.

5. Click 'save' when finished.

9.2 Creating Connections

At this point, you can have one or more source groups and one or more tool destinations.



172-31-60-77 

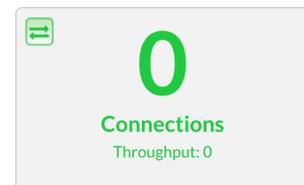
 172-31-54-54

1. To connect a source group to a destination/tool, simply click on the source and drag the connection line to the required destination and drop it. This will popup the following connection profile window, with the 'Source Group' and 'Destination Group' values preset.

Add Connection ✕

<h4 style="margin: 0;">Select Source Group</h4> <p>Source Group *</p> <div style="background-color: #007bff; color: white; padding: 5px; text-align: center; margin-top: 5px;">172-31-60-77</div>	<h4 style="margin: 0;">Define Connection</h4> <p>Connection Type * VNI</p> <p><input checked="" type="radio"/> VXLAN <input type="radio"/> GRE ID</p> <p>Description</p> <div style="border: 1px solid #ccc; height: 40px; margin-top: 5px;"></div> <p>Define Services</p> <p>Berkeley Packet Filter (BPF) ?</p> <div style="border: 1px solid #ccc; padding: 2px; margin-top: 5px;">Type here...</div>	<h4 style="margin: 0;">Select Destination</h4> <p>Destination Group *</p> <div style="background-color: #6f42c1; color: white; padding: 5px; text-align: center; margin-top: 5px;">172-31-54-54</div>
---	---	---

Save



2. You can also click on the the icon in top left corner of on the Connections box and it will take you to the same screen, but you will have to choose the Source Group and Destination Group from their drop-down.

Add Connection
✕

Select Source Group

Source Group *

Source Group...

Define Connection

Connection Type *

VXLAN GRE

VNI ID

Description

Description...

Define Services

Berkeley Packet Filter (BPF) ?

Type here...

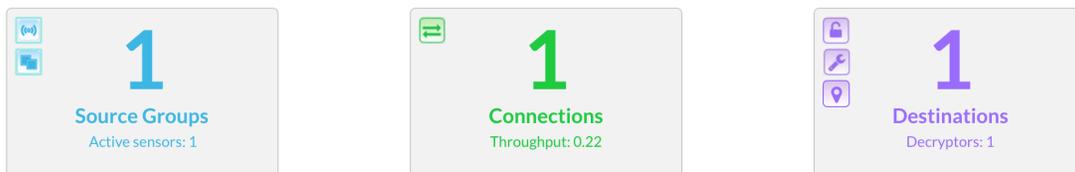
Select Destination

Destination Group *

Destination Group...

Save

3. You can use VXLAN tunnel encapsulation to send traffic to the destination. The VNI ID should be a unique number for the source group.
4. Berkeley Packet Filter (*BPF*) is where you enter the data filters you want to apply to this connection.
5. Click save to return to the dashboard. You will see mirroring indications in the diagram as well as the ‘throughput’ value in the connections box:



172-31-60-77



172-31-54-54

If you generate encrypted traffic on your source instance:

```
# run some https traffic on the client
curl https://example.com
```

You can see the decrypted traffic on your destination by issuing the command on the decryptor instance:

```
tcpdump -Ani nurx0 port 80
```

9.3 Decrypting AWS VPC Traffic Mirroring

To decrypt Amazon VPC traffic mirroring follow the first three steps described above:

1. [Installing Nubeva Sensors](#)
2. [Creating Source Groups](#)
3. [Launching Decryptors](#)

Since the AWS VPC traffic mirrors are generating the packet flow, there is no need to create connections. Instead, setup an AWS VPC traffic mirroring session between your source where the nubeva sensor is running, and the destination where you installed the decryptor engine.

To set up a traffic mirroring session please review [. Additional information is available on the .](#)

9.4 Monitoring Sensors and Decryptors

You can monitor the status of your `Sensors` and `Decryptors` on the `sensors` and `decryptors` pages respectively. The figure below depicts the layout of the `sensors` page showing one active sensor:

Active Sensors

Activity Threshold Show Active Inactive All Session key learning is enabled using private Nubeva hosted key depot

Instance ID	Cloud	Region	IPV4	Private Hostname	Operating System	Unique ID (suffix)	Uptime (hr) Last Heard	Source Group(s)	KBps	
i-07dcb7d23f66ca52b	AWS	us-east-1	172.31.60.77	ip-172-31-60-77.ec2.internal	Linux	4956325528	99 (hr) 1/14 17:31:35	172-31-60-77	0.28	

Active Decryptors

Activity Threshold Show Active Inactive All Session key learning is enabled using private Nubeva hosted key depot

Instance ID	Cloud	Region	IPV4	Private Hostname	Operating System	Unique ID	Uptime (hr) Last Heard	
i-0e4af2e0f0d4d1696	AWS	us-east-1	172.31.54.54	ip-172-31-54-54.ec2.internal	Linux	9863085458	99 (hr) 01/14 17:34:48	

Both screens allow you to set the ‘sensitivity threshold’ for determining whether a sensor or a decryptor is ‘active’. Active states are define by the time since a sensor/decryptor last ‘phoned home’. The two figures below show how the same sensor is considered inactive with a threshold set to 6 hours and active when the threshold is set to 1 day.

Sensor active during the past 24 hours

Home **Sensors** Decryptors Help

Activity Threshold Show Active Inactive All Session key learning is enabled using a private key depot

Instance ID	Cloud	Region	IPV4	Private Hostname	Operating System	Unique ID (suffix)	Uptime (hr) Last Heard	Source Group(s)	KBps	
i-0b64b6fdf40520df6	AWS	us-east-1	172.31.52.75	ip-172-31-52-75.ec2.internal	Linux	5112798106	109.2 (hr) 1/08 04:15:14	SG 75	7.72	

Sensor inactive in the past six hours

Activity Threshold **6 Hours**

Show **Active** Inactive All

Session key learning is enabled using a private key depot

Instance ID	Cloud	Region	IPV4	Private Hostname	Operating System	Unique ID (suffix)	Uptime (hr) Last Heard	Source Group(s)	KBps	
i-0b64b6fdf40520df6	AWS	us-east-1	172.31.52.75	ip-172-31-52-75.ec2.internal	Linux	5112798106	109.2 (hr) 1/08 04:15:14	SG 75	7.72	

CHAPTER 10

TLS API

Please follow the link to Nubeva's TLS

Deploying Security Tools

This guide shows how to add five open-source security & networking tools to your AWS Cloud environment. It deploys Moloch, ntopng, Suricata, Wireshark, and Zeek that provide advanced security visibility in your cloud. These tools are integrated with the Nubeva TLS Decryption solution with provides deeper visibility into all TLS traffic, including TLS 1.2 w/PFS and TLS 1.3. You can choose to create a new VPC environment for your open-source tools or deploy them into your existing VPC environment. After you deploy the Quick Start, you can add other AWS services, infrastructure components, and software layers to complete your test.

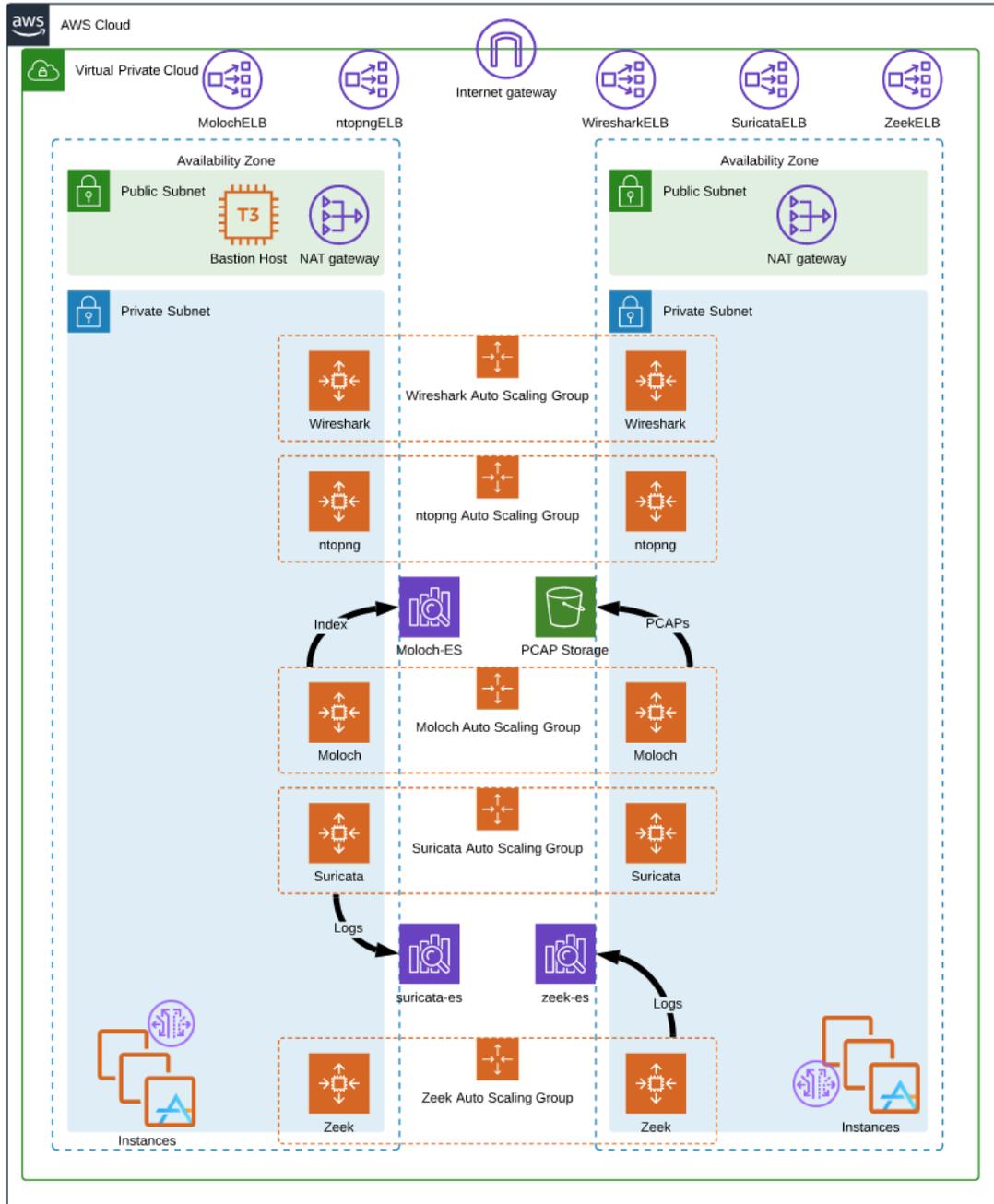
11.1 Tool Launcher

Please follow the steps below to launch cloud tools.



1. Click on the “Wrench” (middle) icon on the top left of the Destination Group box. This will display the popup depicted in the figure below.

AWS/NUBEVA QUICKSTART ARCHITECTURE



2. Select the region and the VPC options.
3. Click the copy button to copy the Nubeva Token to the clipboard. This token is required by the cloud formation template that orchestrates tool launches.
4. Click the Launch Tool button. This loads a cloud formation template that allows you to select which tools you would like to launch.

The following sections describe each tool deployment in more detail.

11.2 Moloch

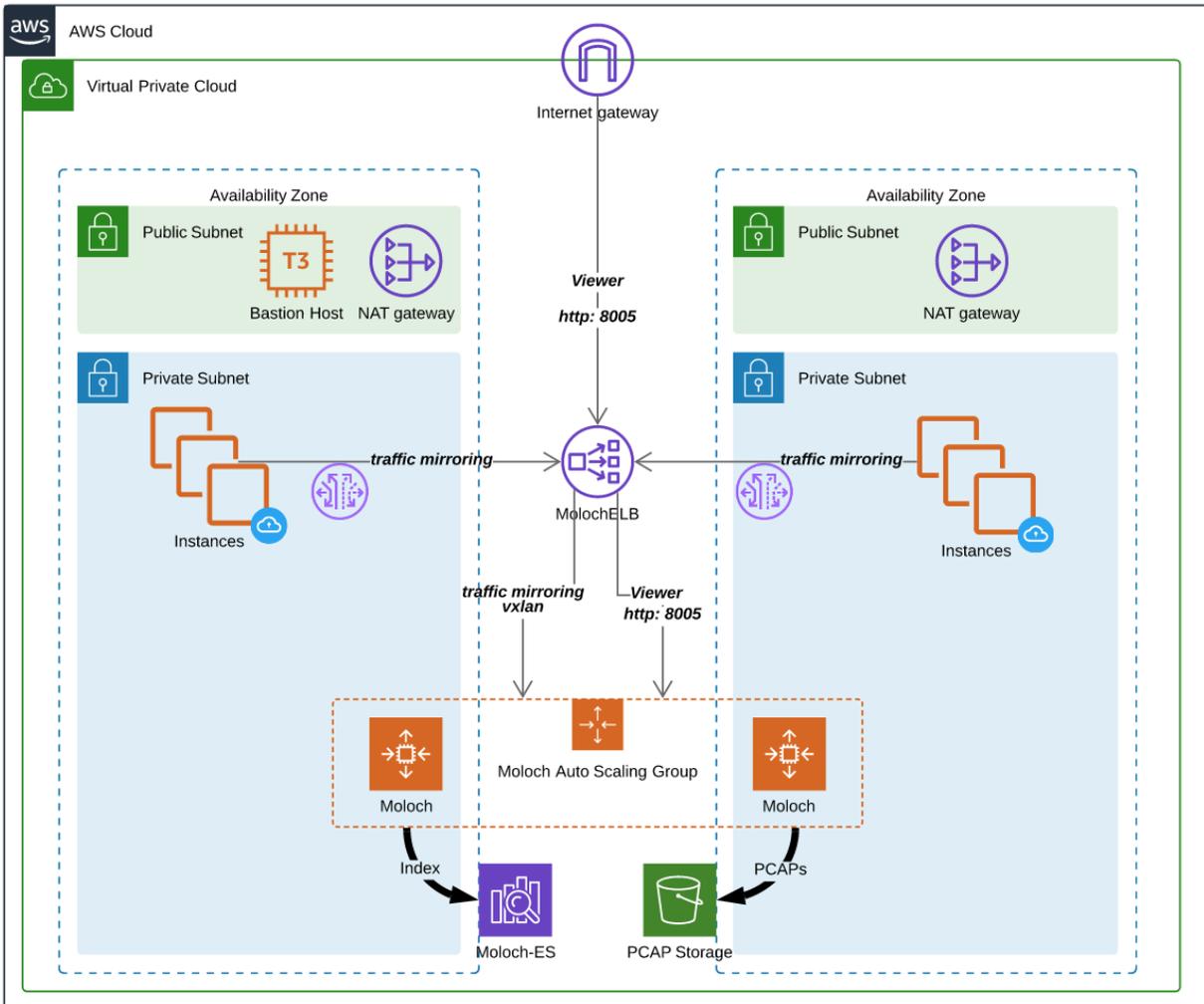
Moloch is a large scale, open-source, indexed packet capture and search system. Moloch augments your current security infrastructure to store and index network traffic in standard PCAP format, providing fast, indexed access. An intuitive and simple web interface is provided for PCAP browsing, searching, and exporting. Moloch exposes APIs which allow for PCAP data and JSON formatted session data to be downloaded and consumed directly. Moloch stores and exports all packets in standard PCAP format, allowing you to also use your favorite PCAP ingesting tools such as Wireshark, during your analysis workflow.

For additional information please refer to [as well as the](#) .

11.2.1 Moloch Architecture Details

As part of the Nubeva Tools automated deployment, Moloch is deployed using an [. The figure below depicts the complete highly scalable Moloch architecture.](#)

NUBEVA TOOLS - MOLOCH ARCHITECTURE AWS



- Moloch EC2 instances are built from code, look at the for more details.
- Each Moloch EC2 instance contains an active Moloch Viewer and Moloch Capture
- Moloch is installed at /data/moloch
- All Moloch components start automatically with the /data/moloch/start_moloch.sh script.
- Moloch logs are located at /data/moloch/logs
- Moloch will be configured to use the username & password which were defined as part of the CFT creation.
- **All Moloch EC2 instances use the AWS Elasticsearch service, moloch-es.**
 - VPC access w/security group restrictions (see below for more info)
 - Uses same machine type for ES cluster nodes
 - Uses same node count for ES cluster nodes
 - Uses port 80/http for all ES communication (can be changed after install to 443/https)
- More Moloch details are in the config file /data/moloch/etc/config.ini.

- Moloch only monitors nurx0
- **Network Elastic Load Balancers front-end all communications to the Moloch instances**
 - UDP port 4789 is forwarded to all targets for Amazon VPC traffic mirrors
 - TCP port 8005 is forwarded to all targets for Moloch Viewer

11.2.2 Operating Moloch

- Connect to the MolochELB on port 8005 using HTTP. Login in with the tooladmin username & password.
- Point all VPC traffic mirroring sessions at the Traffic Mirror Target (TMT) for the MolochELB. This will ensure that the mirrors are sent to any active Moloch capture point. This leverages UDP load balancing, so flows will stick on the Moloch capture engines.
- All Moloch instances will then store packet information in the AWS ElasticSearch Service that is created during the CFT process.
- The final traffic PCAPs are stored in the S3 bucket created during the CFT process.
- To view the moloch data, use a web browser to connect to the load balancer URL on port 8005.

11.2.3 Moloch Security Details

- Each Moloch EC2 instance allows TCP port 22 (ssh) and 8005 (Moloch viewer using http) from the Remote Access CIDR specified at the CFT launch.
- Each Moloch EC2 instance allows UDP port 4789 (vxlan) from any source in the VPC.
- Each Moloch EC2 instance has unlimited outbound access
- The AWS ElasticSearch service allow TCP port 80 (http) from any source in the VPC.

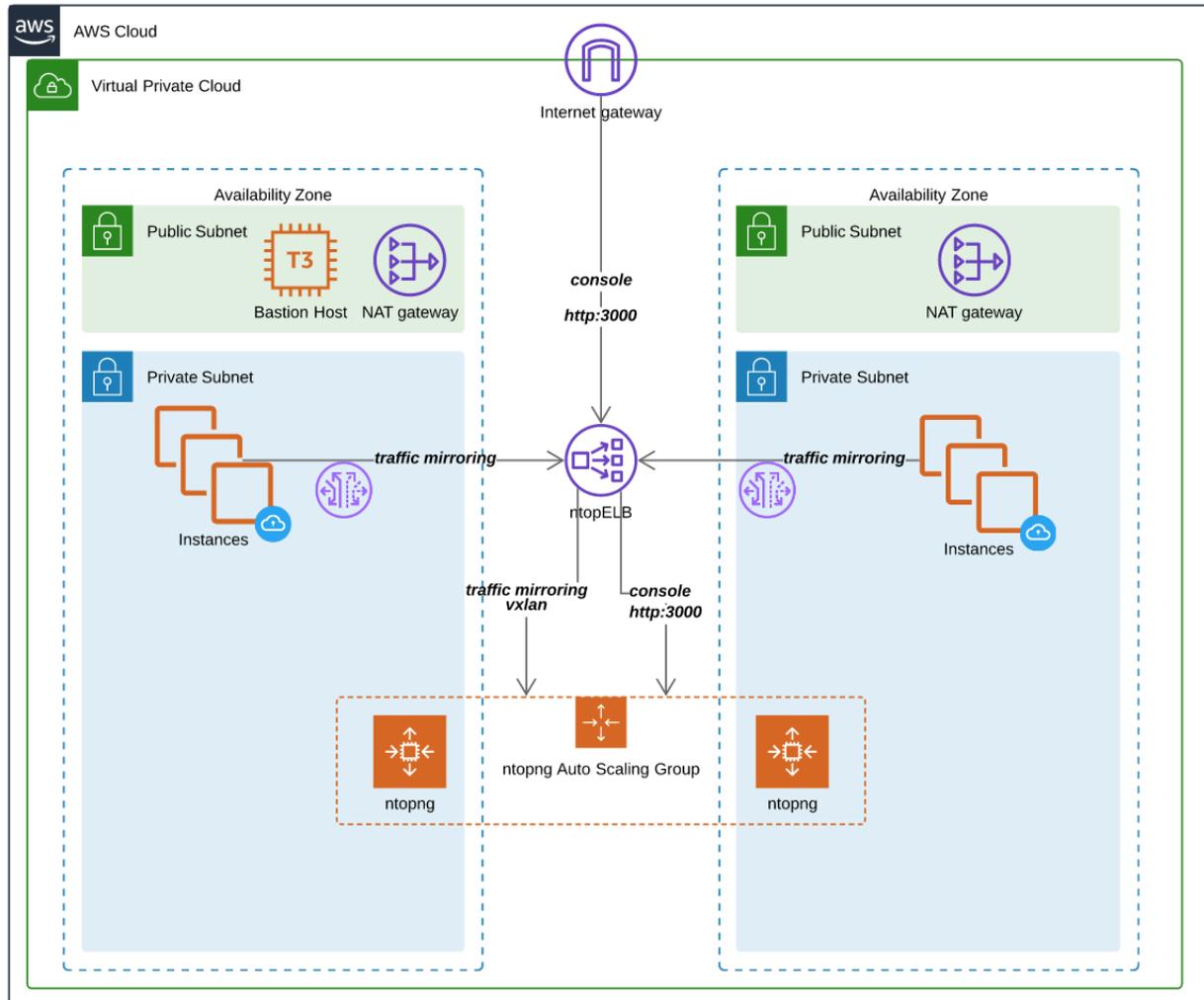
11.3 ntopng

ntopng is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. For more details refer to .

11.3.1 ntopng Architecture Details

As part of the Nubeva Tools automated deployment, ntopng is deployed using an . The figure below depicts the complete highly scalable Suricata architecture.

NUBEVA TOOLS - NTOPNG ARCHIECTURE AWS



- ntopng instances are built from code, look at the for more details.
- ntopng is installed using yum and uses all the defaults
- ntopng only monitors nurx0.
- Any browser can be used to connect to the ntopng console on port 3000 if they are part of the Remote Access CIDR. .
- **Network Elastic Load Balancers front-end all communications to the ntopng instances**
 - UDP port 4789 is forwarded to all targets for Amazon VPC traffic mirrors
 - TCP port 3000 is forwarded to all targets for incoming console connections.

11.3.2 Operating ntopng

- Connect to the NtopELB on port 3000 using HTTP. Login in with the default username & password for ntop
- Point all VPC traffic mirroring sessions at the Traffic Mirror Target (TMT) for the ntopELB. This will ensure that the mirrors are sent to any active ntopng instance. This leverages UDP load balancing, so flows will stick

on the ntopng instance.

- To view the ntopng UI, connect to the load balancer URL on port 3000.

11.3.3 ntopng Security Details

- Each ntopng EC2 instance allows TCP port 22 (ssh) and 3000 (ntopng console using http) from the Remote Access CIDR specified at the CFT launch.
- Each ntopng EC2 instance allows UDP port 4789 (vxlan) from any source in the VPC.
- Each ntopng EC2 instance has unlimited outbound access

11.4 Suricata

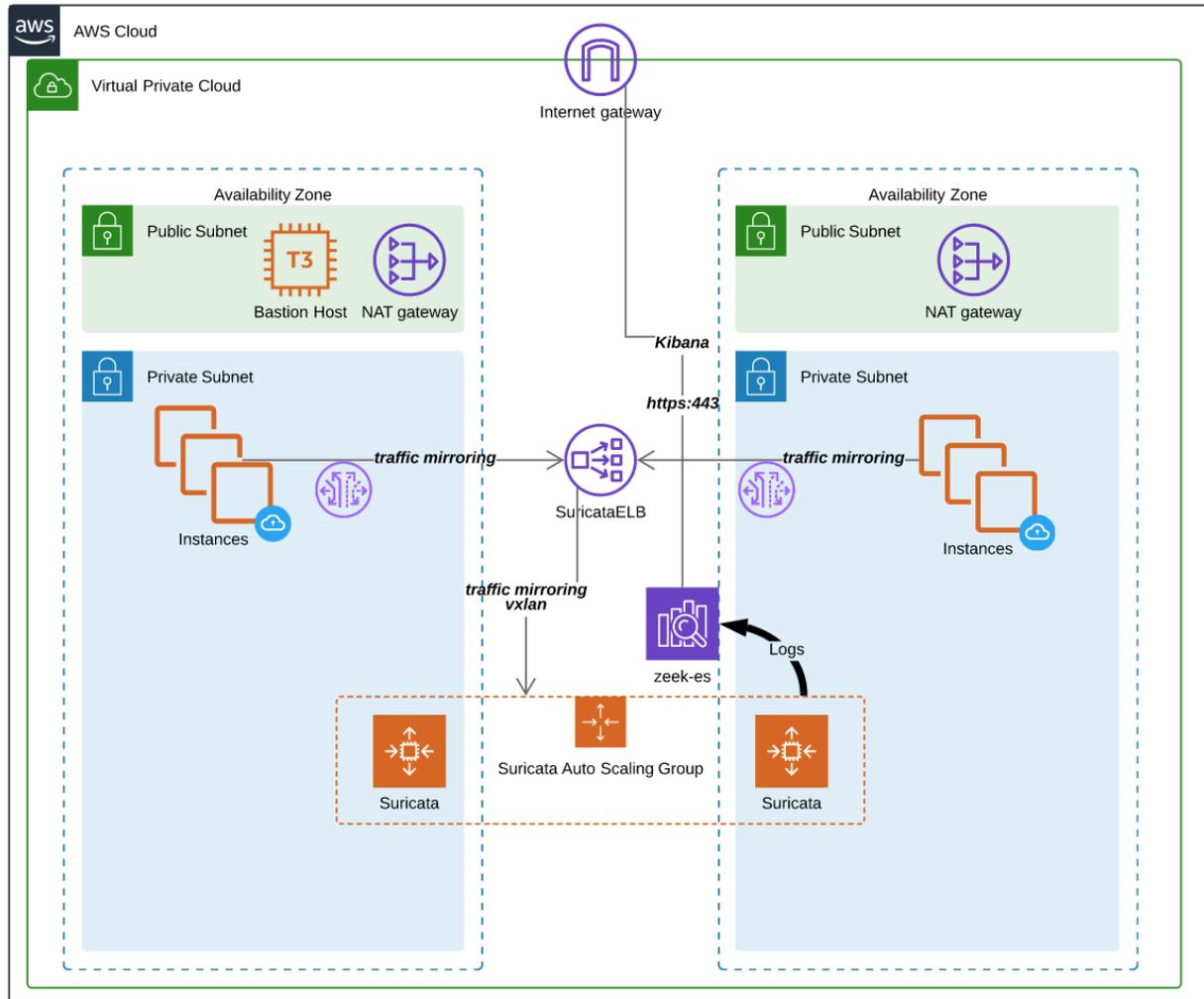
Suricata is a high performance Network IDS, IPS and Network Security Monitoring engine. It is open-source and owned by a community-run non-profit foundation, the Open Information Security Foundation (OISF). Suricata is developed by the OISF.

For additional information please refer to [as well as the](#) .

11.4.1 Suricata Architecture Details

As part of the Nubeva Tools automated deployment, Suricata is deployed using an [. The figure below depicts the complete highly scalable Suricata architecture.](#)

NUBEVA TOOLS - SURICATA ARCHIECTURE AWS



- Suricata EC2 instances are built from code, look at the for more details.
- Each Suricata EC2 instance contains an active Suricata worker and Logstash for log storage.
- Suricata is installed via yum with all defaults
- All Suricata components start automatically as a service.
- Suricata logs are located at `/var/log/suricata/`
- Suricata only monitors `nurx0`
- **All Suricata EC2 instances use the AWS Elasticsearch service, `suricata-es`.**
 - VPC access w/security group restrictions (see below for more info)
 - Uses same machine type for ES cluster nodes
 - Uses same node count for ES cluster nodes
 - Uses port 80/http for all ES communication (can be changed after install to 443/https)
- More Suricata details are in the config file `/etc/suricata/suricata.yaml`

- Suricatas alerts can be viewed through the Kibana UI integrated with the AWS ES service. See the output of the Suricata CFT for the exact URL.
- **Network Elastic Load Balancers front-end all communications to the Suricata instances**
 - UDP port 4789 is forwarded to all targets for Amazon VPC traffic mirrors

11.4.2 Operating Suricata

- Connect to the Kibana link in the Suricata CFT Output section for access.
- Point all VPC traffic mirroring sessions at the Traffic Mirror Target (TMT) for the SuricataELB. This will ensure that the mirrors are sent to any active Suricata worker. This leverages UDP load balancing, so flows will stick on the Suricata workers.
- All Suricata logs are sent to the AWS ElasticSearch Service that is created during the CFT process.
- To view the Suricata data, use a web browser to connect to the Kibana URL specified in the output of the Suricata CFT.

11.4.3 Suricata Security Details

- Each Suricata EC2 instance allows TCP port 22 (ssh) from the Remote Access CIDR specified at the CFT launch.
- Each Suricata EC2 instance allows UDP port 4789 (vxlan) from any source in the VPC.
- Each Suricata EC2 instance has unlimited outbound access
- The AWS ElasticSearch service allow TCP port 80 (http) from any source in the VPC and TCP port 443 (https) from the Remote Access CIDR specified at launch.

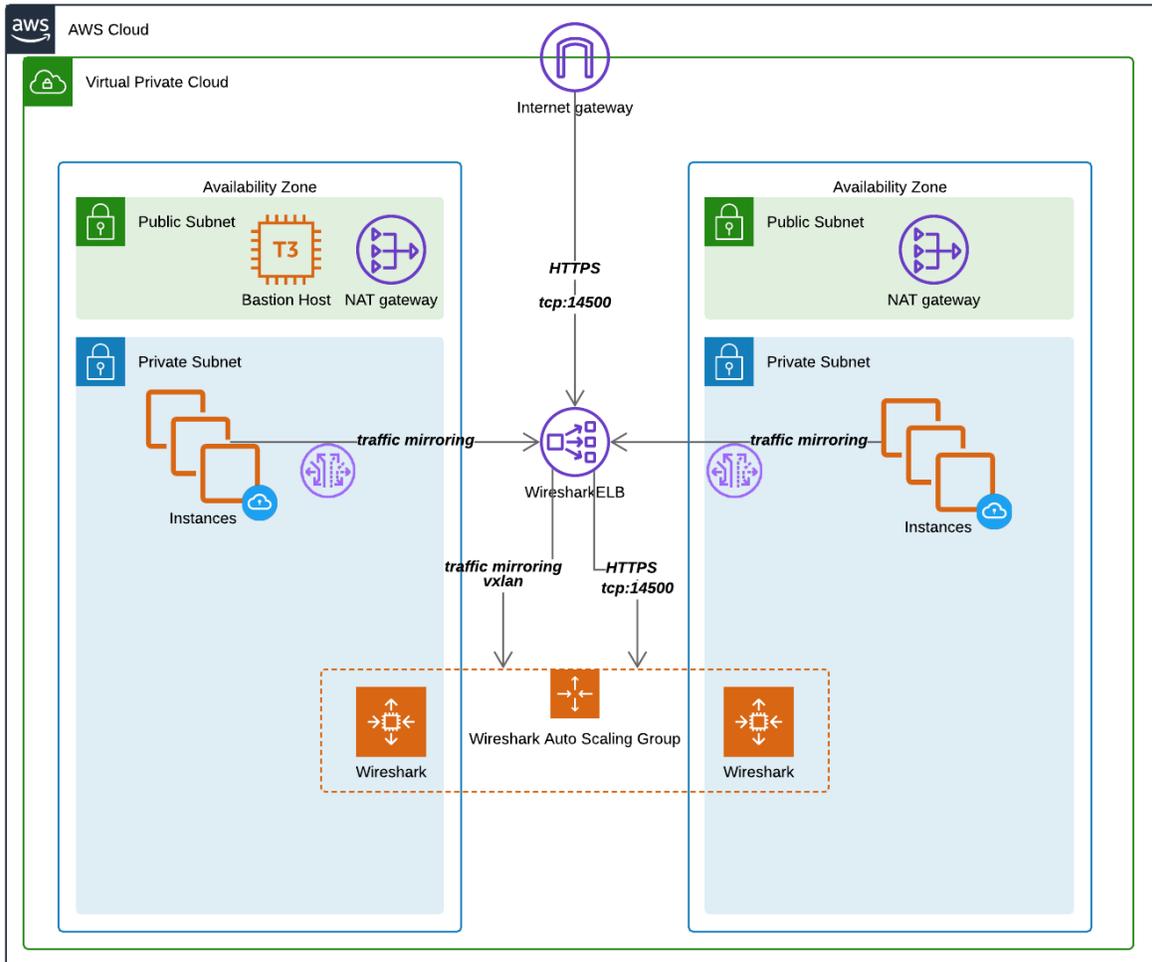
11.5 Wireshark

Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. for more information refer to .

11.5.1 Wireshark Architecture Details

As part of the Nubeva Tools automated deployment, Wireshark is deployed using an . The figure below depicts the complete highly scalable Suricata architecture.

NUBEVA TOOLS - WIRESHARK ARCHIECTURE AWS



- Wireshark instances are built from code, look at the for more details.
- Each Wireshark EC2 instance contains a Wireshark docker container accessed using HTTPS.
- Select the nurx0 interface to see the decapsulated and decrypted packets.
- Network Elastic Load Balancers front-end all communications to the Wireshark instances
- UDP port 4789 is forwarded to all targets for Amazon VPC traffic mirrors
- TCP port 14500 is forwarded to all targets for incoming HTTPS connections.

11.5.2 Operating Wireshark

- Connect to the WiresharkhELB on port 14500 using HTTPS. Login in with the tooladmin username & password.
- Point all VPC traffic mirroring sessions at the Traffic Mirror Target (TMT) for the WiresharkELB. This will ensure that the mirrors are sent to any active Wireshark instance. This leverages UDP load balancing, so flows will stick on the Wireshark instance.

- To view the wireshark UI, use any browser to connect to the load balancer URL, located on the console or in the output of the CFT.

11.5.3 Wireshark Security Details

- Each Wireshark EC2 instance allows TCP port 22 (ssh) and 14500 (HTTPS) from the Remote Access CIDR specified at the CFT launch.
- Each Wireshark EC2 instance allows UDP port 4789 (vxlan) from any source in the VPC.
- Each Wireshark EC2 instance has unlimited outbound access

11.6 Zeek

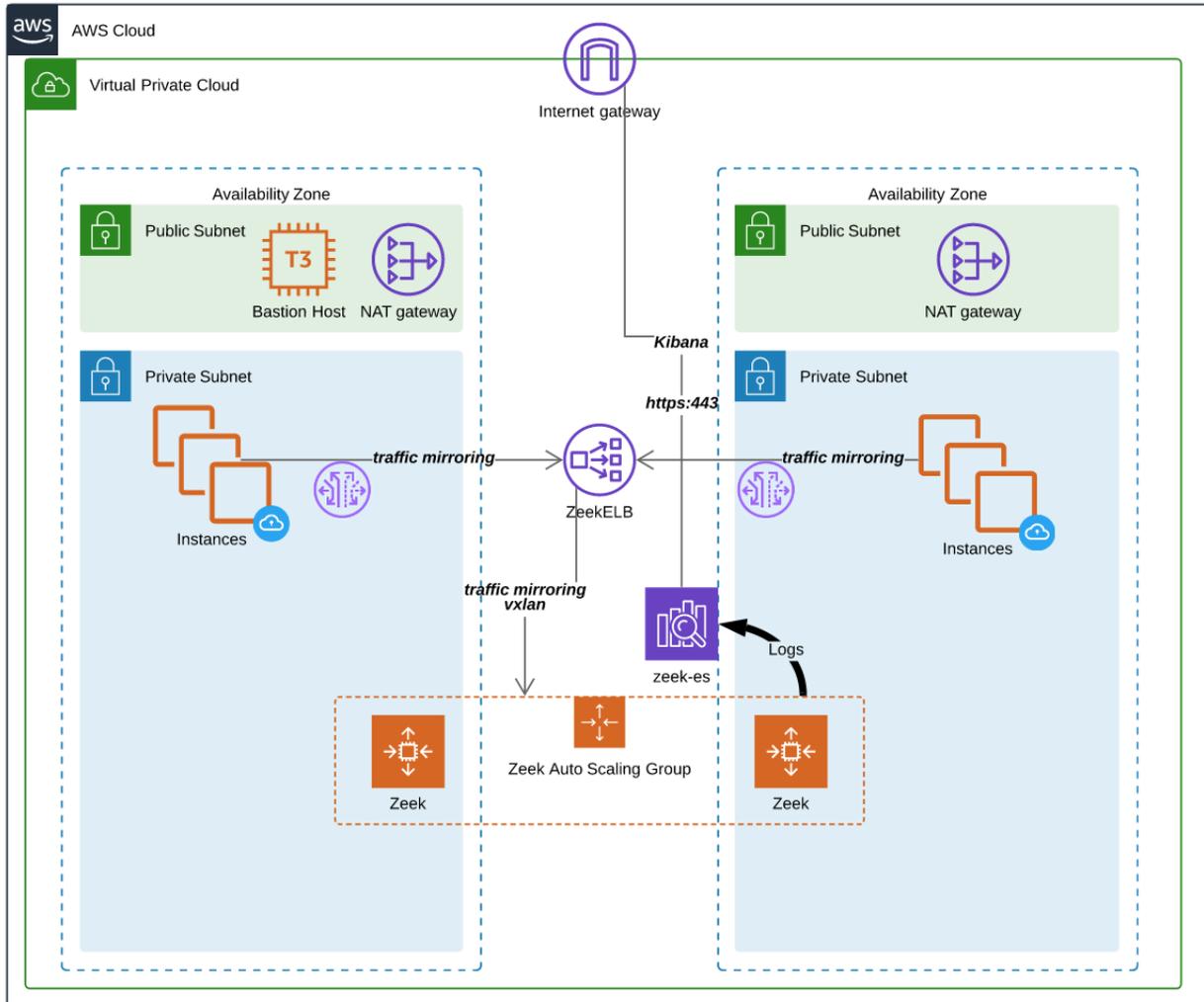
Zeek is a powerful network analysis framework that is much different from the typical IDS you may know. While focusing on network security monitoring, Zeek provides a comprehensive platform for more general network traffic analysis as well. Well grounded in more than 20 years of research, Zeek has successfully bridged the traditional gap between academia and operations since its inception. Today, it is relied upon operationally by both major companies and numerous educational and scientific institutions for securing their cyberinfrastructure.

For additional information please refer to as well as the .

11.6.1 Zeek Architecture Details

As part of the Nubeva Tools automated deployment, Zeek is deployed using an . The figure below depicts the complete highly scalable Suricata architecture.

NUBEVA TOOLS - ZEEK ARCHIECTURE AWS



- Zeek EC2 instances are built from code, look at the for more details.
- Each Zeek EC2 instance contains an active Zeek worker and Logstash for log storage.
- Zeek is installed at `/opt/zeek`
- All Zeek components start automatically with the `/opt/zeek/start_zeek.sh` script.
- Zeek logs are located at `/opt/zeek/logs`
- Zeek only monitors `nurx0`
- All Zeek EC2 instances use the AWS ElasticSearch service, `zeek-es`.
- VPC access w/security group restrictions (see below for more info)
- Uses same machine type for ES cluster nodes
- Uses same node count for ES cluster nodes
- Uses port `80/http` for all ES communication (can be changed after install to `443/https`)

- Zeeks alerts can be viewed through the Kibana UI integrated with the AWS ES service. See the output of the Zeek CFT for the exact URL.
- Network Elastic Load Balancers front-end all communications to the Zeek instances
- UDP port 4789 is forwarded to all targets for Amazon VPC traffic mirrors

11.6.2 Operating Zeek

- Connect to the Kibana link in the Zeek CTF Output section for access.
- Point all VPC traffic mirroring sessions at the Traffic Mirror Target (TMT) for the ZeekELB. This will ensure that the mirrors are sent to any active Zeek worker. This leverages UDP load balancing, so flows will stick on the Zeek workers.
- All Zeek logs are sent to the AWS ElasticSearch Service that is created during the CFT process.
- To view the zeek data, use a web browser to connect to the Kibana URL specified in the output of the Zeek CFT..

11.6.3 Zeek Security Details

- Each Zeek EC2 instance allows TCP port 22 (ssh) from the Remote Access CIDR specified at the CFT launch.
- Each Zeek EC2 instance allows UDP port 4789 (vxlan) from any source in the VPC.
- Each Zeek EC2 instance has unlimited outbound access
- The AWS ElasticSearch service allow TCP port 80 (http) from any source in the VPC and TCP port 443 (https) from the Remote Access CIDR specified at launch.

CHAPTER 12

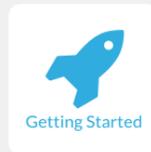
Help and Support

To report issues or if there is anything else you would like to see in the product, please let us know. Click on the Help page and fill in the form. That will tell us directly what you think we should do next.

 NUBEVA	Project Default Project	User yiftah.porat@gmail.co	
Home	Sensors	Decryptors	Help

Need Help?

Here Are Some Resources



Submit a Request

Please fill out the form below. A support ticket will be generated and the Nubeva Team will be happy to assist you!

Subject

How can we help?

Attach Screenshot (Optional)

Click to upload an image

Frequently Asked Questions

Question: What measures & research has Nubeva undertaken to ensure that your products continue to operate given the following challenges:

- OS vendor changes
- Compliance issues
- Interaction with AV/Malware/EDR products
- TPM-type solutions
- Protocol updates, changes and additions

13.1 OS Vendor Changes

Today on Microsoft Windows, Nubeva uses ‘hooking’ for Symmetric Key Intercept. That means we act, with permission, in the user space application process and memory space. So whatever mechanism segmentation functions, OS or hardware, Nubeva acts from the user-space process perspective to get the memory segment that has the keys.

For almost twenty years, ‘hooking’ has been licensed by hundreds of ISVs, used by nearly every product team at Microsoft, and is a generally accepted method.

Microsoft Windows is moving towards tracing, having joined Open DTrace. Windows itself had kernel tracing mechanisms such as ETW already. But is now choosing to invest further into more advanced tracing methods such as DTrace. Once tracing mechanisms are widely available in all Windows distributions, Nubeva looks forward to using tracing instead of hooking.

Nubeva already uses tracing inside the Linux operating system to access process memory. Just like in Windows hooking, Nubeva accesses the memory just as the process does. The biggest difference with tracing is that the tracing works directly in the operating system and is first validated as safe, unobtrusive, and not overly complex by the OS kernel. This method is safer and more performant than hooking. That is why it’s expected that all operating systems are likely to evolve to support similar systems.

13.2 Government/Compliance Issues

Nubeva has been classified as EAR99. Software classified as EAR99 does not require any additional licenses for export. Nubeva Sensors do not participate in encryption processes in any way. Therefore there are no additional requirements or considerations required for export.

13.3 Interaction with AV/Malware/EDR Products

Nubeva sensors do not trigger an alert from products such as Windows defenders and other AV/Malware/EDR products in their default form. However, if the products change from their defaults and have all alerts turned on, then some will detect Nubeva's hooking. They all have an easy method to add the Nubeva process to the "allowed processes list". This requirement goes hand in glove with the concept that Nubeva is a solution that must be explicitly given permissions to operate by administrators.

13.4 TPM-type solutions

Trusted Platform Modules (TPM) are used to secure long-term asymmetric encryption and authentication secrets, such as certificates, private keys, and passwords. Microsoft Pluton further secures TPMs to protect the communication between the TPM and CPU so it cannot be intercepted. Neither TPM nor Pluton affects Nubeva. Nubeva looks for ephemeral keys from the user space process memory. The Nubeva process does not depend on any asymmetric keys nor the TPM architecture. By acting exactly as the user process would (DTrace on Linux, hooking in Windows), Nubeva can access the key from memory exactly as the user process does. The user process ultimately needs to access the symmetric key given that TLS is an application encryption protocol. Not network layer protocol like IPSEC. So it must have access to symmetric keys

13.5 Protocol Updates, Changes & Additions

Nubeva operations have a complete set of automated tests to detect new versions of software, protocols, and applications. As new versions are detected, Nubeva automatically creates new signatures and tests for viability. If key extraction is successful, then the new signatures are automatically pushed to the master repo as well as to all partners. If key extraction is not successful, then the Nubeva R&D team manually creates the new signatures and extends the testing process. The automated signature creation takes 60-90 minutes to complete and if manual intervention is required, another 2-4 hours is required on average for a new signature.

For applications and libraries that we have source code access to, for instance, open-source, it generally takes us a day or two to apply our intellectual property to extract the keys. Nubeva also tests the preview version of software to ensure we can support them at launch.

CHAPTER 14

Installing Docker

For general information about docker please visit

Key Agents require version 18.09 of Docker which might not be the default version in your cloud. The instructions below show how to install versions of docker which our agents support.

14.1 Ubuntu 18.04

```
apt update && apt install -y docker.io
```

14.2 AWS Linux

AWS Linux AMI version 1 cannot install 18.09 the following commands will install 18.06.1-ce which works correctly.

```
On a completely new instance:  
  
Amazon Linux AMI  
  
sudo yum update -y  
sudo yum upgrade  
sudo yum install -y docker  
sudo usermod -aG docker $USER  
exit  
# ssh back to instance  
sudo service docker start
```

14.3 AWS Linux 2

On a completely new instance:

```
Amazon Linux 2 AMI (HVM), SSD Volume Type
amzn2-ami-hvm-2.0.20190508-x86_64-gp2 (ami-0cb72367e98845d43)

sudo yum update -y
sudo amazon-linux-extras install docker
sudo service docker start
sudo usermod -a -G docker ec2-user
exit
```

14.4 Cent OS 7

```
yum install -y docker
yum remove -y docker docker-common docker-selinux docker-engine
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
yum install -y docker-ce
systemctl enable docker.service
systemctl start docker.service
```

14.5 Red Hat Enterprise Linux 7.5 (RHEL)

```
yum update -y
yum-config-manager --enable rhui-REGION-rhel-server-extras
yum install -y container-selinux yum-utils
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
yum install -y docker-ce
systemctl enable docker.service
systemctl start docker.service
```

14.6 Cloud Provider Instructions

Note: The cloud provider default version of Docker might not be compatible with our agents

These are the links to install cloud provider versions of docker on [AWS](#) and [Azure](#) instances.

CHAPTER 15

Ciphers Supported

"TLSv1.3"			
TLS_AES_256_GCM_SHA384	Kx=any	Au=Any	Enc=AESGCM(256) Mac=AEAD
TLS_CHACHA20_POLY1305_SHA256	Kx=any	Au=Any	Enc=CHACHA20/POLY1305(256) ↵
↵Mac=AEAD			
TLS_AES_128_GCM_SHA256	Kx=Any	Au=Any	Enc=AESGCM(128) Mac=AEAD
"TLSv1.2"			
ECDHE-ECDSA-AES256-GCM-SHA384	Kx=ECDH	Au=ECDSA	Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-GCM-SHA384	Kx=ECDH	Au=RSA	Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-GCM-SHA384	Kx=DH	Au=RSA	Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-CHACHA20-POLY1305	Kx=ECDH	Au=ECDSA	Enc=CHACHA20/POLY1305(256) ↵
↵Mac=AEAD			
ECDHE-RSA-CHACHA20-POLY1305	Kx=ECDH	Au=RSA	Enc=CHACHA20/POLY1305(256) ↵
↵Mac=AEAD			
DHE-RSA-CHACHA20-POLY1305	Kx=DH	Au=RSA	Enc=CHACHA20/POLY1305(256) ↵
↵Mac=AEAD			
ECDHE-ECDSA-AES128-GCM-SHA256	Kx=ECDH	Au=ECDSA	Enc=AESGCM(128) Mac=AEAD
ECDHE-RSA-AES128-GCM-SHA256	Kx=ECDH	Au=RSA	Enc=AESGCM(128) Mac=AEAD
DHE-RSA-AES128-GCM-SHA256	Kx=DH	Au=RSA	Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES256-SHA384	Kx=ECDH	Au=ECDSA	Enc=AES(256) Mac=SHA384
ECDHE-RSA-AES256-SHA384	Kx=ECDH	Au=RSA	Enc=AES(256) Mac=SHA384
DHE-RSA-AES256-SHA256	Kx=DH	Au=RSA	Enc=AES(256) Mac=SHA256
ECDHE-ECDSA-AES128-SHA256	Kx=ECDH	Au=ECDSA	Enc=AES(128) Mac=SHA256
ECDHE-RSA-AES128-SHA256	Kx=ECDH	Au=RSA	Enc=AES(128) Mac=SHA256
DHE-RSA-AES128-SHA256	Kx=DH	Au=RSA	Enc=AES(128) Mac=SHA256
AES256-GCM-SHA384	Kx=RSA	Au=RSA	Enc=AESGCM(256) Mac=AEAD
AES128-GCM-SHA256	Kx=RSA	Au=RSA	Enc=AESGCM(128) Mac=AEAD
AES256-SHA256	Kx=RSA	Au=RSA	Enc=AES(256) Mac=SHA256
AES128-SHA256	Kx=RSA	Au=RSA	Enc=AES(128) Mac=SHA256
ECDHE-PSK-CHACHA20-POLY1305			
↵Mac=AEAD	Kx=ECDHEPSK	Au=PSK	Enc=CHACHA20/POLY1305(256) ↵
RSA-PSK-AES256-GCM-SHA384			
↵Mac=AEAD	Kx=RSAPSK	Au=RSA	Enc=AESGCM(256) Mac=AEAD
DHE-PSK-AES256-GCM-SHA384			
↵Mac=AEAD	Kx=DHEPSK	Au=PSK	Enc=AESGCM(256) Mac=AEAD

(continues on next page)

(continued from previous page)

RSA-PSK-CHACHA20-POLY1305 ↔Mac=AEAD	Kx=RSAPSK	Au=RSA	Enc=CHACHA20/POLY1305 (256)	↵
DHE-PSK-CHACHA20-POLY1305 ↔Mac=AEAD	Kx=DHEPSK	Au=PSK	Enc=CHACHA20/POLY1305 (256)	↵
RSA-PSK-AES128-GCM-SHA256	Kx=RSAPSK	Au=RSA	Enc=AESGCM (128)	Mac=AEAD
DHE-PSK-AES128-GCM-SHA256	Kx=DHEPSK	Au=PSK	Enc=AESGCM (128)	Mac=AEAD
"TLSv1 "				
ECDHE-ECDSA-AES256-SHA	Kx=ECDH	Au=ECDSA	Enc=AES (256)	Mac=SHA1
ECDHE-RSA-AES256-SHA	Kx=ECDH	Au=RSA	Enc=AES (256)	Mac=SHA1
ECDHE-ECDSA-AES128-SHA	Kx=ECDH	Au=ECDSA	Enc=AES (128)	Mac=SHA1
ECDHE-RSA-AES128-SHA	Kx=ECDH	Au=RSA	Enc=AES (128)	Mac=SHA1
ECDHE-PSK-AES256-CBC-SHA384	Kx=ECDHEPSK	Au=PSK	Enc=AES (256)	Mac=SHA384
ECDHE-PSK-AES256-CBC-SHA	Kx=ECDHEPSK	Au=PSK	Enc=AES (256)	Mac=SHA1
RSA-PSK-AES256-CBC-SHA384	Kx=RSAPSK	Au=RSA	Enc=AES (256)	Mac=SHA384
DHE-PSK-AES256-CBC-SHA384	Kx=DHEPSK	Au=PSK	Enc=AES (256)	Mac=SHA384
ECDHE-PSK-AES128-CBC-SHA256	Kx=ECDHEPSK	Au=PSK	Enc=AES (128)	Mac=SHA256
ECDHE-PSK-AES128-CBC-SHA	Kx=ECDHEPSK	Au=PSK	Enc=AES (128)	Mac=SHA1
RSA-PSK-AES128-CBC-SHA256	Kx=RSAPSK	Au=RSA	Enc=AES (128)	Mac=SHA256
DHE-PSK-AES128-CBC-SHA256	Kx=DHEPSK	Au=PSK	Enc=AES (128)	Mac=SHA256

Supported Libraries and Operating Systems

16.1 OpenSSL

```
OpenSSL-0.9.7
OpenSSL-0.9.7-beta4
OpenSSL-0.9.7-beta5
OpenSSL-0.9.7-beta5
OpenSSL-0.9.7-beta6
OpenSSL-0.9.7a
OpenSSL-0.9.7b
OpenSSL-0.9.7c
OpenSSL-0.9.7d
OpenSSL-0.9.7e
OpenSSL-0.9.7f
OpenSSL-0.9.7g
OpenSSL-0.9.7h
OpenSSL-0.9.7i
OpenSSL-0.9.7j
OpenSSL-0.9.7k
OpenSSL-0.9.7l
OpenSSL-0.9.7m
OpenSSL-0.9.8
OpenSSL-0.9.8-beta1
OpenSSL-0.9.8-beta2
OpenSSL-0.9.8-beta3
OpenSSL-0.9.8-beta4
OpenSSL-0.9.8-beta5
OpenSSL-0.9.8-beta6
OpenSSL-0.9.8-post-auto-reformat
OpenSSL-0.9.8-post-reformat
OpenSSL-0.9.8-pre-auto-reformat
OpenSSL-0.9.8-pre-reformat
OpenSSL-0.9.8a
OpenSSL-0.9.8b
```

(continues on next page)

(continued from previous page)

```
OpenSSL-0.9.8c
OpenSSL-0.9.8d
OpenSSL-0.9.8e
OpenSSL-0.9.8f
OpenSSL-0.9.8g
OpenSSL-0.9.8h
OpenSSL-0.9.8i
OpenSSL-0.9.8j
OpenSSL-0.9.8k
OpenSSL-0.9.8l
OpenSSL-0.9.8m
OpenSSL-0.9.8m-beta1
OpenSSL-0.9.8n
OpenSSL-0.9.8o
OpenSSL-0.9.8p
OpenSSL-0.9.8q
OpenSSL-0.9.8r
OpenSSL-0.9.8s
OpenSSL-0.9.8t
OpenSSL-0.9.8u
OpenSSL-0.9.8v
OpenSSL-0.9.8w
OpenSSL-0.9.8x
OpenSSL-0.9.8y
OpenSSL-0.9.8za
OpenSSL-0.9.8zb
OpenSSL-0.9.8zc
OpenSSL-0.9.8zd
OpenSSL-0.9.8ze
OpenSSL-0.9.8zf
OpenSSL-0.9.8zg
OpenSSL-0.9.8zh
OpenSSL-1.0.0
OpenSSL-1.0.0-beta1
OpenSSL-1.0.0-beta2
OpenSSL-1.0.0-beta3
OpenSSL-1.0.0-beta4
OpenSSL-1.0.0-beta5
OpenSSL-1.0.0-post-auto-reformat
OpenSSL-1.0.0-post-reformat
OpenSSL-1.0.0-pre-auto-reformat
OpenSSL-1.0.0-pre-reformat
OpenSSL-1.0.0a
OpenSSL-1.0.0b
OpenSSL-1.0.0c
OpenSSL-1.0.0d
OpenSSL-1.0.0e
OpenSSL-1.0.0f
OpenSSL-1.0.0g
OpenSSL-1.0.0h
OpenSSL-1.0.0i
OpenSSL-1.0.0j
OpenSSL-1.0.0k
OpenSSL-1.0.0l
OpenSSL-1.0.0m
OpenSSL-1.0.0n
OpenSSL-1.0.0o
```

(continues on next page)

(continued from previous page)

```
OpenSSL-1.0.0p
OpenSSL-1.0.0q
OpenSSL-1.0.0r
OpenSSL-1.0.0s
OpenSSL-1.0.0t
OpenSSL-1.0.1
OpenSSL-1.0.1-beta1
OpenSSL-1.0.1-beta2
OpenSSL-1.0.1-beta3
OpenSSL-1.0.1-post-auto-reformat
OpenSSL-1.0.1-post-reformat
OpenSSL-1.0.1-pre-auto-reformat
OpenSSL-1.0.1-pre-reformat
OpenSSL-1.0.1a
OpenSSL-1.0.1b
OpenSSL-1.0.1c
OpenSSL-1.0.1d
OpenSSL-1.0.1e
OpenSSL-1.0.1f
OpenSSL-1.0.1g
OpenSSL-1.0.1h
OpenSSL-1.0.1i
OpenSSL-1.0.1j
OpenSSL-1.0.1k
OpenSSL-1.0.1l
OpenSSL-1.0.1m
OpenSSL-1.0.1n
OpenSSL-1.0.1o
OpenSSL-1.0.1p
OpenSSL-1.0.1q
OpenSSL-1.0.1r
OpenSSL-1.0.1s
OpenSSL-1.0.1t
OpenSSL-1.0.1u
OpenSSL-1.0.2
OpenSSL-1.0.2-beta1
OpenSSL-1.0.2-beta1-fips
OpenSSL-1.0.2-beta2
OpenSSL-1.0.2-beta2-fips
OpenSSL-1.0.2-beta3
OpenSSL-1.0.2-beta3-fips
OpenSSL-1.0.2-fips
OpenSSL-1.0.2-post-auto-reformat
OpenSSL-1.0.2-post-auto-reformat-fips
OpenSSL-1.0.2-post-reformat
OpenSSL-1.0.2-post-reformat-fips
OpenSSL-1.0.2-pre-auto-reformat
OpenSSL-1.0.2-pre-auto-reformat-fips
OpenSSL-1.0.2-pre-reformat
OpenSSL-1.0.2-pre-reformat-fips
OpenSSL-1.0.2a
OpenSSL-1.0.2a-fips
OpenSSL-1.0.2b
OpenSSL-1.0.2b-fips
OpenSSL-1.0.2c
OpenSSL-1.0.2c-fips
OpenSSL-1.0.2d
```

(continues on next page)

(continued from previous page)

```
OpenSSL-1.0.2d-fips
OpenSSL-1.0.2e
OpenSSL-1.0.2e-fips
OpenSSL-1.0.2f
OpenSSL-1.0.2f-fips
OpenSSL-1.0.2g
OpenSSL-1.0.2g-fips
OpenSSL-1.0.2h
OpenSSL-1.0.2h-fips
OpenSSL-1.0.2i
OpenSSL-1.0.2i-fips
OpenSSL-1.0.2j
OpenSSL-1.0.2j-fips
OpenSSL-1.0.2k
OpenSSL-1.0.2k-fips
OpenSSL-1.0.2l
OpenSSL-1.0.2l-fips
OpenSSL-1.0.2m
OpenSSL-1.0.2m-fips
OpenSSL-1.0.2n
OpenSSL-1.0.2n-fips
OpenSSL-1.0.2o
OpenSSL-1.0.2o-fips
OpenSSL-1.0.2p
OpenSSL-1.0.2p-fips
OpenSSL-1.0.2q
OpenSSL-1.0.2q-fips
OpenSSL-1.0.2r
OpenSSL-1.0.2r-fips
OpenSSL-1.0.2s
OpenSSL-1.0.2s-fips
OpenSSL-1.0.2t
OpenSSL-1.0.2t-fips
OpenSSL-1.0.2u
OpenSSL-1.0.2u-fips
OpenSSL-1.1.0
OpenSSL-1.1.0-pre1
OpenSSL-1.1.0-pre2
OpenSSL-1.1.0-pre3
OpenSSL-1.1.0-pre4
OpenSSL-1.1.0-pre5
OpenSSL-1.1.0-pre6
OpenSSL-1.1.0a
OpenSSL-1.1.0b
OpenSSL-1.1.0c
OpenSSL-1.1.0d
OpenSSL-1.1.0e
OpenSSL-1.1.0f
OpenSSL-1.1.0g
OpenSSL-1.1.0h
OpenSSL-1.1.0i
OpenSSL-1.1.0j
OpenSSL-1.1.0k
OpenSSL-1.1.0l
OpenSSL-1.1.1
OpenSSL-1.1.1-pre1
OpenSSL-1.1.1-pre2
```

(continues on next page)

(continued from previous page)

```
OpenSSL-1.1.1-pre3
OpenSSL-1.1.1-pre4
OpenSSL-1.1.1-pre5
OpenSSL-1.1.1-pre6
OpenSSL-1.1.1-pre7
OpenSSL-1.1.1-pre8
OpenSSL-1.1.1-pre9
OpenSSL-1.1.1a
OpenSSL-1.1.1b
OpenSSL-1.1.1c
OpenSSL-1.1.1d
OpenSSL-1.1.1e
OpenSSL-1.1.1f
OpenSSL-1.1.1g
OpenSSL-1.1.1h
OpenSSL-1.1.1i
OpenSSL-FIPS.1.0
OpenSSL-fips-1.2.0
OpenSSL-fips-1.2.1
OpenSSL-fips-1.2.2
OpenSSL-fips-1.2.3
OpenSSL-fips-2.0
OpenSSL-fips-2.0-pl1
OpenSSL-fips-2.0-rc1
OpenSSL-fips-2.0-rc2
OpenSSL-fips-2.0-rc3
OpenSSL-fips-2.0-rc4
OpenSSL-fips-2.0-rc5
OpenSSL-fips-2.0-rc6
OpenSSL-fips-2.0-rc7
OpenSSL-fips-2.0-rc8
OpenSSL-fips-2.0-rc9
OpenSSL-fips-2.0.1
OpenSSL-fips-2.0.2
OpenSSL-fips-2.0.3
OpenSSL-fips-2.0.4
OpenSSL-fips-2.0.5
OpenSSL-fips-2.0.6
OpenSSL-fips-2.0.7
OpenSSL-fips-2.0.8
OpenSSL-fips-2.0.9
OpenSSL-fips-2.0.10
OpenSSL-fips-2.0.11
OpenSSL-fips-2.0.12
OpenSSL-fips-2.0.13
OpenSSL-fips-2.0.14
OpenSSL-fips-2.0.15
OpenSSL-fips-2.0.16
```

16.2 NSS Libraries

Every version greater than v3.15 (released 2013-05-28 23:37:46)

NSS 3.15
NSS 3.15.1
NSS 3.15.2
NSS 3.15.3
NSS 3.15.3.1
NSS 3.15.4
NSS 3.15.5
NSS 3.16
NSS 3.16.1
NSS 3.16.2
NSS 3.16.2.1
NSS 3.16.2.2
NSS 3.16.2.3
NSS 3.16.3
NSS 3.16.4
NSS 3.16.5
NSS 3.16.6
NSS 3.17
NSS 3.17.1
NSS 3.17.2
NSS 3.17.3
NSS 3.17.4
NSS 3.18
NSS 3.18.1
NSS 3.19
NSS 3.19.1
NSS 3.19.2
NSS 3.19.3
NSS 3.20
NSS 3.20.1
NSS 3.21
NSS 3.21.1
NSS 3.21.2
NSS 3.21.3
NSS 3.21.4
NSS 3.22
NSS 3.22.1
NSS 3.22.2
NSS 3.23
NSS 3.24
NSS 3.25
NSS 3.25.1
NSS 3.26
NSS 3.26.2
NSS 3.27
NSS 3.27.1
NSS 3.27.2
NSS 3.28
NSS 3.28.1
NSS 3.28.2
NSS 3.28.3
NSS 3.28.4
NSS 3.28.5
NSS 3.29
NSS 3.29.1
NSS 3.29.2
NSS 3.29.3

(continues on next page)

(continued from previous page)

NSS 3.29.5
NSS 3.30
NSS 3.30.1
NSS 3.30.2
NSS 3.31
NSS 3.31.1
NSS 3.32
NSS 3.33
NSS 3.34
NSS 3.34.1
NSS 3.35
NSS 3.36
NSS 3.36.1
NSS 3.36.2
NSS 3.36.4
NSS 3.36.5
NSS 3.37
NSS 3.37.1
NSS 3.37.3
NSS 3.38
NSS 3.39
NSS 3.40
NSS 3.36.6
NSS 3.40.1
NSS 3.41
NSS 3.36.7
NSS 3.36.8
NSS 3.42
NSS 3.42.1
NSS 3.43
NSS 3.44
NSS 3.44.1
NSS 3.44.2
NSS 3.44.3
NSS 3.45
NSS 3.46
NSS 3.46.1
NSS 3.47
NSS 3.47.1
NSS 3.48
NSS 3.48.1
NSS 3.49
NSS 3.49.1
NSS 3.49.2
NSS 3.50
NSS 3.51
NSS 3.51.1
NSS 3.52
NSS 3.53
NSS 3.54
NSS 3.55

16.3 WolfSSL

Version v4.3.0

16.4 Linux

Nubeva TLS sensors are supported on Ubuntu, RHEL, CentOS and AWS AMI.

16.5 MS Windows

Nubeva TLS sensors are supported on Windows Server 2012, Server 2012 R2, Server 2016, Server 2019 and Windows 10.

Supported applications on these Microsoft platforms:

Ap- pli- ca- tion	Version
Ac- cess	All
Cor- tana	All
Drop- box	91.4.548, 92.4.382, 94.4.384, 95.4.441, 99.4.501, 100.4.409, 101.4.434, 102.4.431, 103.4.383, 104.4.175, 108.4.453, 110.4.458, 111.4.472, 112.4.321, 113.4.507
Excel	All
Groove Mu- sic	All
Google Chrome	80.0.3987.132, 80.0.3987.149, 81.0.4044.113, 83.0.4103.97, 83.0.4103.10, 83.0.4103.116, 84.0.4147.89, 84.0.4174.105, 84.0.4147.125, 85.0.4183.83, 85.0.4183.102, 85.0.4183.121, 86.0.4240.75, 86.0.4240.183, 86.0.4240.193, 86.0.4240.198, 87.0.4280.66, 87.0.4280.88, 87.0.4280.141, 88.0.4324.104
In- foPath	All
Inter- net Ex- plorer	All
Mi- crosoft Store	All
Mi- crosoft News	All
Mi- crosoft Sway	All
Mi- crosoft Teams	All
Mi- crosoft Word	All
Movies & TV	All
MS Dy- nam- ics 365 CRM	All
MS Edge (old)	44.18362.449.0
MS Edge Chromium	80.0.361.66, 80.0.361.69, 80.0.361.109, 83.0.478.44, 83.0.478.45, 83.0.478.54, 83.0.478.56, 83.0.478.58, 83.0.478.61, 84.0.522.44, 84.0.522.49, 84.0.522.52, 84.0.522.59, 84.0.522.63, 85.0.564.41, 85.0.564.44, 85.0.564.51, 85.0.564.63, 86.0.4240.75, 86.0.622.38, 86.0.622.43, 86.0.622.51, 86.0.622.56, 86.0.622.61, 86.0.622.69, 87.0.664.41, 87.0.664.47, 87.0.664.52, 87.0.664.55, 87.0.664.57, 87.0.664.60, 87.0.664.66, 87.0.664.75, 88.0.705.50
MSN Money	All
16.5. MS Windows	
MSN Sports	All
MSN	All

Schannel.dll versions supported:

```
6.1.7601.17514
6.2.17763.802
6.2.9200.22562
6.3.9600.17415
6.3.9600.19473
10.0.14393.1613
10.0.14393.3269
10.0.14393.3750
10.0.14393.3808
10.0.14393.3930
10.0.17763.1
10.0.17763.1217
10.0.17763.1282
10.0.17763.1339
10.0.17763.1457
10.0.17763.802
10.0.18362.1082
10.0.18362.418
10.0.18362.900
10.0.18362.959
10.0.18362.997
10.0.19041.1
10.0.19041.329
10.0.19041.388
10.0.19041.508
10.0.19041.546
```

Berkeley Packet Filters

Berkeley Packet Filter (BPF) Syntax

The *expression* consists of one or more *primitives*. Primitives usually consist of an *id* (name or number) preceded by one or more qualifiers. There are three different kinds of qualifier:

type qualifiers say what kind of thing the *id* name or number refers to. Possible types are **host**, **net**, **port** and **portrange**. E.g., ‘host foo’, ‘net 128.3’, ‘port 20’, ‘portrange 6000-6008’. If there is no type qualifier, **host** is assumed.

dir qualifiers specify a particular transfer direction to and/or from *id*. Possible directions are **src**, **dst**, **src or dst** and **src and dst**. E.g., ‘src foo’, ‘dst net 128.3’, ‘src or dst port ftp-data’. If there is no *dir* qualifier, **src or dst** is assumed. For some link layers, such as SLIP and the “cooked” Linux capture mode used for the “any” device and for some other device types, the **inbound** and **outbound** qualifiers can be used to specify a desired direction.

proto qualifiers restrict the match to a particular protocol. Possible protos are: **ether**, **fddi**, **tr**, **wlan**, **ip**, **ip6**, **arp**, **rarp**, **decnet**, **tcp** and **udp**. E.g., ‘ether src foo’, ‘arp net 128.3’, ‘tcp port 21’, ‘udp portrange 7000-7009’. If there is no *proto* qualifier, all protocols consistent with the type are assumed. E.g., ‘src foo’ means ‘(ip or arp or rarp) src foo’ (except the latter is not legal syntax), ‘net bar’ means ‘(ip or arp or rarp) net bar’ and ‘port 53’ means ‘(tcp or udp) port 53’.

‘fddi’ is actually an alias for ‘ether’; the parser treats them identically as meaning “the data link level used on the specified network interface.” FDDI headers contain Ethernet-like source and destination addresses, and often contain Ethernet-like packet types, so you can filter on these FDDI fields just as with the analogous Ethernet fields. FDDI headers also contain other fields, but you cannot name them explicitly in a filter expression.

Similarly, ‘tr’ and ‘wlan’ are aliases for ‘ether’; the previous paragraph’s statements about FDDI headers also apply to Token Ring and 802.11 wireless LAN headers. For 802.11 headers, the destination address is the DA field and the source address is the SA field; the BSSID, RA, and TA fields aren’t tested.

In addition to the above, there are some special ‘primitive’ keywords that don’t follow the pattern: **gateway**, **broadcast**, **less**, **greater** and arithmetic expressions. All of these are described below.

More complex filter expressions are built up by using the words **and**, **or** and **not** to combine primitives. E.g., ‘host foo and not port ftp and not port ftp-data’. To save typing, identical qualifier lists can be omitted. E.g., ‘tcp dst port ftp or ftp-data or domain’ is exactly the same as ‘tcp dst port ftp or tcp dst port ftp-data or tcp dst port domain’.

Allowable primitives are:

dst host *host* True if the IPv4/v6 destination field of the packet is *host*, which may be either an address or a name.

src host *host* True if the IPv4/v6 source field of the packet is *host*.

host *host* True if either the IPv4/v6 source or destination of the packet is *host*. Any of the above host expressions can be prepended with the keywords, **ip**, **arp**, **rarp**, or **ip6** as in:

```
ip host host
```

which is equivalent to:

```
ether proto \ip and host host
```

If *host* is a name with multiple IP addresses, each address will be checked for a match.

ether dst *ehost* True if the Ethernet destination address is *ehost*. *Ehost* may be either a name from /etc/ethers or a number (see '*ethers* </cgi-bin/man/man2html?5+ethers>' __ (5) for numeric format).

ether src *ehost* True if the Ethernet source address is *ehost*.

ether host *ehost* True if either the Ethernet source or destination address is *ehost*.

gateway *host* True if the packet used *host* as a gateway. I.e., the Ethernet source or destination address was *host* but neither the IP source nor the IP destination was *host*. *Host* must be a name and must be found both by the machine's host-name-to-IP-address resolution mechanisms (host name file, DNS, NIS, etc.) and by the machine's host-name-to-Ethernet-address resolution mechanism (/etc/ethers, etc.). (An equivalent expression is

```
ether host ehost and not host host
```

which can be used with either names or numbers for *host* / *ehost*.) This syntax does not work in IPv6-enabled configuration at this moment.

dst net *net* True if the IPv4/v6 destination address of the packet has a network number of *net*. *Net* may be either a name from the networks database (/etc/networks, etc.) or a network number. An IPv4 network number can be written as a dotted quad (e.g., 192.168.1.0), dotted triple (e.g., 192.168.1), dotted pair (e.g., 172.16), or single number (e.g., 10); the netmask is 255.255.255.255 for a dotted quad (which means that it's really a host match), 255.255.255.0 for a dotted triple, 255.255.0.0 for a dotted pair, or 255.0.0.0 for a single number. An IPv6 network number must be written out fully; the netmask is ff:ff:ff:ff:ff:ff:ff:ff, so IPv6 "network" matches are really always host matches, and a network match requires a netmask length.

src net *net* True if the IPv4/v6 source address of the packet has a network number of *net*.

net *net* True if either the IPv4/v6 source or destination address of the packet has a network number of *net*.

net net mask *netmask* True if the IPv4 address matches *net* with the specific *netmask*. May be qualified with **src** or **dst**. Note that this syntax is not valid for IPv6 *net*.

net net len True if the IPv4/v6 address matches *net* with a netmask *len* bits wide. May be qualified with **src** or **dst**.

dst port *port* True if the packet is ip/tcp, ip/udp, ip6/tcp or ip6/udp and has a destination port value of *port*. The *port* can be a number or a name used in /etc/services (see *tcp(7)* and *udp(7)*). If a name is used, both the port number and protocol are checked. If a number or ambiguous name is used, only the port number is checked (e.g., **dst port 513** will print both tcp/login traffic and udp/who traffic, and **port domain** will print both tcp/domain and udp/domain traffic).

src port *port* True if the packet has a source port value of *port*.

port *port* True if either the source or destination port of the packet is *port*.

dst portrange *port1-port2* True if the packet is ip/tcp, ip/udp, ip6/tcp or ip6/udp and has a destination port value between *port1* and *port2*. *port1* and *port2* are interpreted in the same fashion as the *port* parameter for **port**.

src portrange *port1-port2* True if the packet has a source port value between *port1* and *port2*.

portrange *port1-port2* True if either the source or destination port of the packet is between *port1* and *port2*. Any of the above port or port range expressions can be prepended with the keywords, **tcp** or **udp**, as in:

```
tcp src port port
```

which matches only tcp packets whose source port is *port*.

less *length* True if the packet has a length less than or equal to *length*. This is equivalent to:

```
len <= length.
```

greater *length* True if the packet has a length greater than or equal to *length*. This is equivalent to:

```
len >= length.
```

ip proto *protocol* True if the packet is an IPv4 packet (see *ip(4P)*) of protocol type *protocol*. *Protocol* can be a number or one of the names **icmp**, **icmp6**, **igmp**, **igrp**, **pim**, **ah**, **esp**, **vrp**, **udp**, or **tcp**. Note that the identifiers **tcp**, **udp**, and **icmp** are also keywords and must be escaped via backslash (\), which is \ in the C-shell. Note that this primitive does not chase the protocol header chain.

ip6 proto *protocol* True if the packet is an IPv6 packet of protocol type *protocol*. Note that this primitive does not chase the protocol header chain.

ip6 protochain *protocol* True if the packet is IPv6 packet, and contains protocol header with type *protocol* in its protocol header chain. For example,

```
ip6 protochain 6
```

matches any IPv6 packet with TCP protocol header in the protocol header chain. The packet may contain, for example, authentication header, routing header, or hop-by-hop option header, between IPv6 header and TCP header. The BPF code emitted by this primitive is complex and cannot be optimized by BPF optimizer code in *tcpdump*, so this can be somewhat slow.

ip protochain *protocol* Equivalent to **ip6 protochain *protocol***, but this is for IPv4.

ether broadcast True if the packet is an Ethernet broadcast packet. The *ether* keyword is optional.

ip broadcast True if the packet is an IPv4 broadcast packet. It checks for both the all-zeroes and all-ones broadcast conventions, and looks up the subnet mask on the interface on which the capture is being done. If the subnet mask of the interface on which the capture is being done is not available, either because the interface on which capture is being done has no netmask or because the capture is being done on the Linux “any” interface, which can capture on more than one interface, this check will not work correctly.

ether multicast True if the packet is an Ethernet multicast packet. The **ether** keyword is optional. This is shorthand for `ether[0] & 1 != 0`.

ip multicast True if the packet is an IPv4 multicast packet.

ip6 multicast True if the packet is an IPv6 multicast packet.

ether proto *protocol* True if the packet is of ether type *protocol*. *Protocol* can be a number or one of the names **ip**, **ip6**, **arp**, **rarp**, **atalk**, **aarp**, **decnet**, **sca**, **lat**, **mopdl**, **moprc**, **iso**, **stp**, **ipx**, or **netbeui**. Note these identifiers are also keywords and must be escaped via backslash (\). [In the case of FDDI (e.g., `‘fddi protocol arp’`), Token Ring (e.g., `‘tr protocol arp’`), and IEEE 802.11 wireless LANS (e.g., `‘wlan protocol arp’`), for most of those protocols, the protocol identification comes from the 802.2 Logical Link Control (LLC) header, which is usually layered on top of the FDDI, Token Ring, or 802.11 header. When filtering for most protocol identifiers on FDDI, Token Ring, or 802.11, *tcpdump* checks only the protocol ID field of an LLC header in so-called SNAP format with an Organizational Unit Identifier (OUI) of 0x000000, for encapsulated Ethernet; it doesn’t check whether the packet is in SNAP format with an OUI of 0x000000. The exceptions are:

iso *tcpdump* checks the DSAP (Destination Service Access Point) and SSAP (Source Service Access Point) fields of the LLC header;

stp and netbeui *tcpdump* checks the DSAP of the LLC header;

atalk *tcpdump* checks for a SNAP-format packet with an OUI of 0x080007 and the AppleTalk etype.

In the case of Ethernet, *tcpdump* checks the Ethernet type field for most of those protocols. The exceptions are:

iso, stp, and netbeui *tcpdump* checks for an 802.3 frame and then checks the LLC header as it does for FDDI, Token Ring, and 802.11;

atalk *tcpdump* checks both for the AppleTalk etype in an Ethernet frame and for a SNAP-format packet as it does for FDDI, Token Ring, and 802.11;

aarp *tcpdump* checks for the AppleTalk ARP etype in either an Ethernet frame or an 802.2 SNAP frame with an OUI of 0x000000;

ipx *tcpdump* checks for the IPX etype in an Ethernet frame, the IPX DSAP in the LLC header, the 802.3-with-no-LLC-header encapsulation of IPX, and the IPX etype in a SNAP frame.

decnet src host True if the DECNET source address is *host*, which may be an address of the form ‘‘10.123’’, or a DECNET host name. [DECNET host name support is only available on ULTRIX systems that are configured to run DECNET.]

decnet dst host True if the DECNET destination address is *host*.

decnet host host True if either the DECNET source or destination address is *host*.

ifname interface True if the packet was logged as coming from the specified interface (applies only to packets logged by OpenBSD’s **pf(4)**).

on interface Synonymous with the **ifname** modifier.

rnr num True if the packet was logged as matching the specified PF rule number (applies only to packets logged by OpenBSD’s **pf(4)**).

rulenum num Synonymous with the **rnr** modifier.

reason code True if the packet was logged with the specified PF reason code. The known codes are: **match**, **bad-offset**, **fragment**, **short**, **normalize**, and **memory** (applies only to packets logged by OpenBSD’s **pf(4)**).

rset name True if the packet was logged as matching the specified PF ruleset name of an anchored ruleset (applies only to packets logged by **pf(4)**).

ruleset name Synonymous with the **rset** modifier.

srnr num True if the packet was logged as matching the specified PF rule number of an anchored ruleset (applies only to packets logged by **pf(4)**).

subrulenum num Synonymous with the **srnr** modifier.

action act True if PF took the specified action when the packet was logged. Known actions are: **pass** and **block** (applies only to packets logged by OpenBSD’s **pf(4)**).

ip, ip6, arp, rarp, atalk, aarp, decnet, **iso, stp, ipx, netbeui** Abbreviations for:

```
ether proto p
```

where *p* is one of the above protocols.

lat, moprc, mopdl Abbreviations for:

```
ether proto p
```

where *p* is one of the above protocols. Note that *tcpdump* does not currently know how to parse these protocols.

vlan [*vlan_id*] True if the packet is an IEEE 802.1Q VLAN packet. If [*vlan_id*] is specified, only true if the packet has the specified *vlan_id*. Note that the first **vlan** keyword encountered in *expression* changes the decoding offsets for the remainder of *expression* on the assumption that the packet is a VLAN packet. The **vlan** [*vlan_id*] expression may be used more than once, to filter on VLAN hierarchies. Each use of that expression increments the filter offsets by 4. For example:

```
vlan 100 && vlan 200
```

filters on VLAN 200 encapsulated within VLAN 100, and

```
vlan && vlan 300 && ip
```

filters IPv4 protocols encapsulated in VLAN 300 encapsulated within any higher order VLAN.

mpls [*label_num*] True if the packet is an MPLS packet. If [*label_num*] is specified, only true if the packet has the specified *label_num*. Note that the first **mpls** keyword encountered in *expression* changes the decoding offsets for the remainder of *expression* on the assumption that the packet is a MPLS-encapsulated IP packet. The **mpls** [*label_num*] expression may be used more than once, to filter on MPLS hierarchies. Each use of that expression increments the filter offsets by 4. For example:

```
mpls 100000 && mpls 1024
```

filters packets with an outer label of 100000 and an inner label of 1024, and

```
mpls && mpls 1024 && host 192.9.200.1
```

filters packets to or from 192.9.200.1 with an inner label of 1024 and any outer label.

pppoed True if the packet is a PPP-over-Ethernet Discovery packet (Ethernet type 0x8863).

pppoes True if the packet is a PPP-over-Ethernet Session packet (Ethernet type 0x8864). Note that the first **pppoes** keyword encountered in *expression* changes the decoding offsets for the remainder of *expression* on the assumption that the packet is a PPPoE session packet. For example:

```
pppoes && ip
```

filters IPv4 protocols encapsulated in PPPoE.

tcp, udp, icmp Abbreviations for:

```
ip proto p or ip6 proto p
```

where *p* is one of the above protocols.

iso proto protocol True if the packet is an OSI packet of protocol type *protocol*. *Protocol* can be a number or one of the names **clnp**, **esls**, or **isis**.

clnp, esls, isis Abbreviations for:

```
iso proto p
```

where *p* is one of the above protocols.

l1, l2, iih, lsp, snp, csnp, psnp Abbreviations for IS-IS PDU types.

vpi n True if the packet is an ATM packet, for SunATM on Solaris, with a virtual path identifier of *n*.

vci n True if the packet is an ATM packet, for SunATM on Solaris, with a virtual channel identifier of *n*.

- lane** True if the packet is an ATM packet, for SunATM on Solaris, and is an ATM LANE packet. Note that the first **lane** keyword encountered in *expression* changes the tests done in the remainder of *expression* on the assumption that the packet is either a LANE emulated Ethernet packet or a LANE LE Control packet. If **lane** isn't specified, the tests are done under the assumption that the packet is an LLC-encapsulated packet.
- llc** True if the packet is an ATM packet, for SunATM on Solaris, and is an LLC-encapsulated packet.
- oamf4s** True if the packet is an ATM packet, for SunATM on Solaris, and is a segment OAM F4 flow cell (VPI=0 & VCI=3).
- oamf4e** True if the packet is an ATM packet, for SunATM on Solaris, and is an end-to-end OAM F4 flow cell (VPI=0 & VCI=4).
- oamf4** True if the packet is an ATM packet, for SunATM on Solaris, and is a segment or end-to-end OAM F4 flow cell (VPI=0 & (VCI=3 | VCI=4)).
- oam** True if the packet is an ATM packet, for SunATM on Solaris, and is a segment or end-to-end OAM F4 flow cell (VPI=0 & (VCI=3 | VCI=4)).
- metac** True if the packet is an ATM packet, for SunATM on Solaris, and is on a meta signaling circuit (VPI=0 & VCI=1).
- bcc** True if the packet is an ATM packet, for SunATM on Solaris, and is on a broadcast signaling circuit (VPI=0 & VCI=2).
- sc** True if the packet is an ATM packet, for SunATM on Solaris, and is on a signaling circuit (VPI=0 & VCI=5).
- ilmic** True if the packet is an ATM packet, for SunATM on Solaris, and is on an ILMI circuit (VPI=0 & VCI=16).
- connectmsg** True if the packet is an ATM packet, for SunATM on Solaris, and is on a signaling circuit and is a Q.2931 Setup, Call Proceeding, Connect, Connect Ack, Release, or Release Done message.
- metaconnect** True if the packet is an ATM packet, for SunATM on Solaris, and is on a meta signaling circuit and is a Q.2931 Setup, Call Proceeding, Connect, Release, or Release Done message.
- expr relop expr** True if the relation holds, where *relop* is one of >, <, >=, <=, =, !=, and *expr* is an arithmetic expression composed of integer constants (expressed in standard C syntax), the normal binary operators [+ , - , * , / , & , | , << , >>], a length operator, and special packet data accessors. Note that all comparisons are unsigned, so that, for example, 0x80000000 and 0xffffffff are > 0. To access data inside the packet, use the following syntax:

```
proto [ expr : size ]
```

Proto is one of **ether**, **fdi**, **tr**, **wlan**, **ppp**, **slip**, **link**, **ip**, **arp**, **rarp**, **tcp**, **udp**, **icmp**, **ip6** or **radio**, and indicates the protocol layer for the index operation. (**ether**, **fdi**, **wlan**, **tr**, **ppp**, **slip** and **link** all refer to the link layer. **radio** refers to the “radio header” added to some 802.11 captures.) Note that *tcp*, *udp* and other upper-layer protocol types only apply to IPv4, not IPv6 (this will be fixed in the future). The byte offset, relative to the indicated protocol layer, is given by *expr*. *Size* is optional and indicates the number of bytes in the field of interest; it can be either one, two, or four, and defaults to one. The length operator, indicated by the keyword **len**, gives the length of the packet.

For example, **ether[0] & 1 != 0** catches all multicast traffic. The expression **ip[0] & 0xf != 5** catches all IPv4 packets with options. The expression **ip[6:2] & 0x1fff = 0** catches only unfragmented IPv4 datagrams and frag zero of fragmented IPv4 datagrams. This check is implicitly applied to the **tcp** and **udp** index operations. For instance, **tcp[0]** always means the first byte of the TCP *header*, and never means the first byte of an intervening fragment.

Some offsets and field values may be expressed as names rather than as numeric values. The following protocol header field offsets are available: **icmptype** (ICMP type field), **icmpcode** (ICMP code field), and **tcpflags** (TCP flags field).

The following ICMP type field values are available: **icmp-echoreply**, **icmp-unreach**, **icmp-sourcequench**, **icmp-redirect**, **icmp-echo**, **icmp-routeradvert**, **icmp-routersolicit**, **icmp-timxceed**, **icmp-paramprob**, **icmp-tstamp**, **icmp-tstampreply**, **icmp-ireq**, **icmp-ireqreply**, **icmp-maskreq**, **icmp-maskreply**.

The following TCP flags field values are available: **tcp-fin**, **tcp-syn**, **tcp-rst**, **tcp-push**, **tcp-ack**, **tcp-urg**.

Primitives may be combined using:

A parenthesized group of primitives and operators (parentheses are special to the Shell and must be escaped). Negation (`**!*` or **not**). Concatenation (`**&&**` or **and**). Alternation (`**||**` or **or**).

Negation has highest precedence. Alternation and concatenation have equal precedence and associate left to right. Note that explicit **and** tokens, not juxtaposition, are now required for concatenation.

If an identifier is given without a keyword, the most recent keyword is assumed. For example,

```
not host vs and ace
```

is short for

```
not host vs and host ace
```

which should not be confused with

```
not ( host vs or ace )
```

Expression arguments can be passed to *tcpdump* as either a single argument or as multiple arguments, whichever is more convenient. Generally, if the expression contains Shell metacharacters, it is easier to pass it as a single, quoted argument. Multiple arguments are concatenated with spaces before being parsed.

EXAMPLES

To capture all packets arriving at or departing from *sundown*:

```
host sundown
```

To capture traffic between *helios* and either *hot* or *ace*:

```
host helios and \( hot or ace \)
```

To capture all IP packets between *ace* and any host except *helios*:

```
ip host ace and not helios
```

To capture all traffic between local hosts and hosts at Berkeley:

```
net ucb-ether
```

To capture all ftp traffic through internet gateway *snupe*: (note that the expression is quoted to prevent the shell from (mis-)interpreting the parentheses):

```
gateway snupe and (port ftp or ftp-data)
```

To capture traffic neither sourced from nor destined for local hosts (if you gateway to one other net, this stuff should never make it onto your local net).

```
ip and not net localnet
```

To capture the start and end packets (the SYN and FIN packets) of each TCP conversation that involves a non-local host.

```
tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net localnet
```

To capture all IPv4 HTTP packets to and from port 80, i.e. print only packets that contain data, not, for example, SYN and FIN packets and ACK-only packets. (IPv6 is left as an exercise for the reader.)

```
tcp port 80 and (((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)
```

To capture IP packets longer than 576 bytes sent through gateway *snoop*:

```
gateway snoop and ip[2:2] > 576
```

To capture IP broadcast or multicast packets that were *not* sent via Ethernet broadcast or multicast:

```
ether[0] & 1 = 0 and ip[16] >= 224
```

To capture all ICMP packets that are not echo requests/replies (i.e., not ping packets):

```
icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply
```

This was taken from the man page of 'tcpdump' <<http://www.tcpdump.org/>> '___.

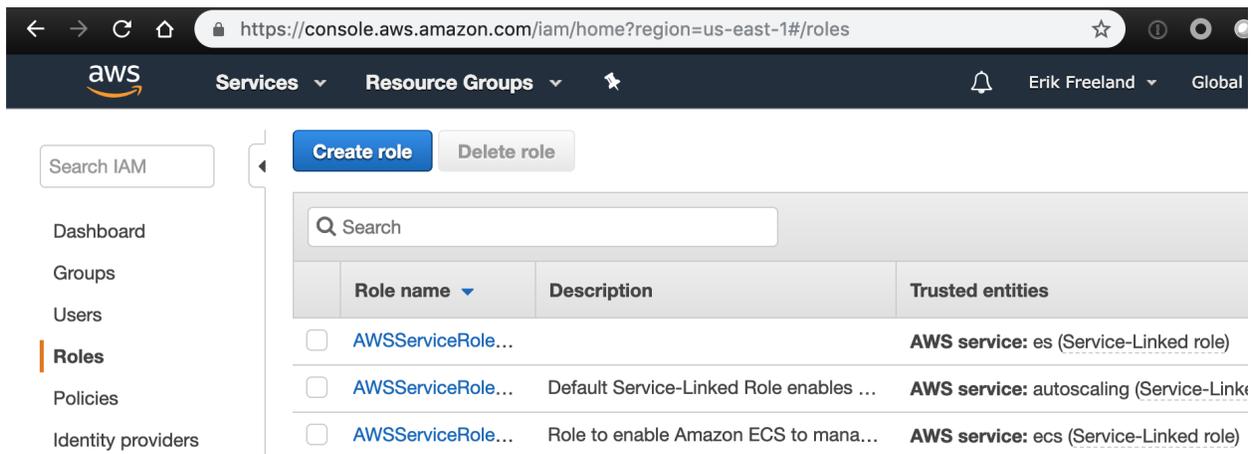
18.1 Using AWS VPC Traffic Mirrors

To set up a traffic mirroring session please review [AWS VPC Traffic Mirrors](#). Additional information is available on the [AWS VPC Traffic Mirrors](#).

18.2 Creating AWS IAM Role for Custom Tag Support

To enable support for AWS Custom Tags, the EC2 instance must have additional AWS permissions that are not enabled by default. The permissions granted by this IAM roles are read-only and will only apply to the EC2 instance itself. The EC2 instance needs permission to read its own tags so it can report them to the SaaS console. The step-by-step instructions for creating these permissions is below.

1. First, go to your AWS Console and select the IAM service. Now click on create role.



2. Choose which service will use the role. This will be EC2. This click the Next:Permissions at the bottom of the page.

- Now you will create a new policy which grants the appropriate permissions. Click on “Create Policy”. This will spawn a new tab, so remember to come back to this tab when creation is complete.

Create role



▼ Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy
↻

Filter policies Showing 499 results

	Policy name	Used as	Description
<input type="checkbox"/>	AdministratorAccess	Permissions policy (1)	Provides full access to AWS services an...
<input type="checkbox"/>	AlexaForBusinessDeviceSetup	None	Provide device setup access to AlexaFor...
<input type="checkbox"/>	AlexaForBusinessFullAccess	None	Grants full access to AlexaForBusiness r...
<input type="checkbox"/>	AlexaForBusinessGatewayExecution	None	Provide gateway execution access to AI...
<input type="checkbox"/>	AlexaForBusinessReadOnlyAccess	None	Provide read only access to AlexaForBu...

- On the create policy screen, select the JSON tab. You will be pasting in the JSON config below. Then select review policy.



Create policy



A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. [Learn more](#)

Visual editor
JSON
[Import managed policy](#)

```

1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "ec2:DescribeInstances"
8       ],
9       "Resource": "*"
10    }
11  ]
12 }
13
    
```

Cancel
Review policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances"
      ],
      "Resource": "*"
    }
  ]
}
```

5. Now give your policy a name and a description; then click create policy.

Create policy



Review policy

Name*

Use alphanumeric and '+=, @-_' characters. Maximum 128 characters.

Description

Maximum 1000 characters. Use alphanumeric and '+=, @-_' characters.

Summary

Service	Access level	Resource	Re
Allow (1 of 171 services) Show remaining 170			
EC2	Limited: List	All resources	No

* Required

[Cancel](#) [Previous](#) [Create policy](#)

6. Remember after creating the policy, go BACK to your Role Creation tab to continue.

7. Now, refresh the policies by clicking the circular arrows on the right. Then search for your newly created policy, here Nubeva-Describe-Instances. Select this then click Next:Tags at the bottom right

Create role



▼ Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy
↻

Filter policies ▾

Showing 1 result

	Policy name ▾	Used as	Description
<input type="checkbox"/>	Nubeva-Describe-Instances	Permissions policy (1)	

8. You can add any tags that you like this will not impact how Nubeva operates or read EC2 instance AWS custom tags.

9. Now give your role a name, and save it.

Create role



Review

Provide the required information below and review this role before you create it.

Role name*

Use alphanumeric and '+=, @-_' characters. Maximum 64 characters.

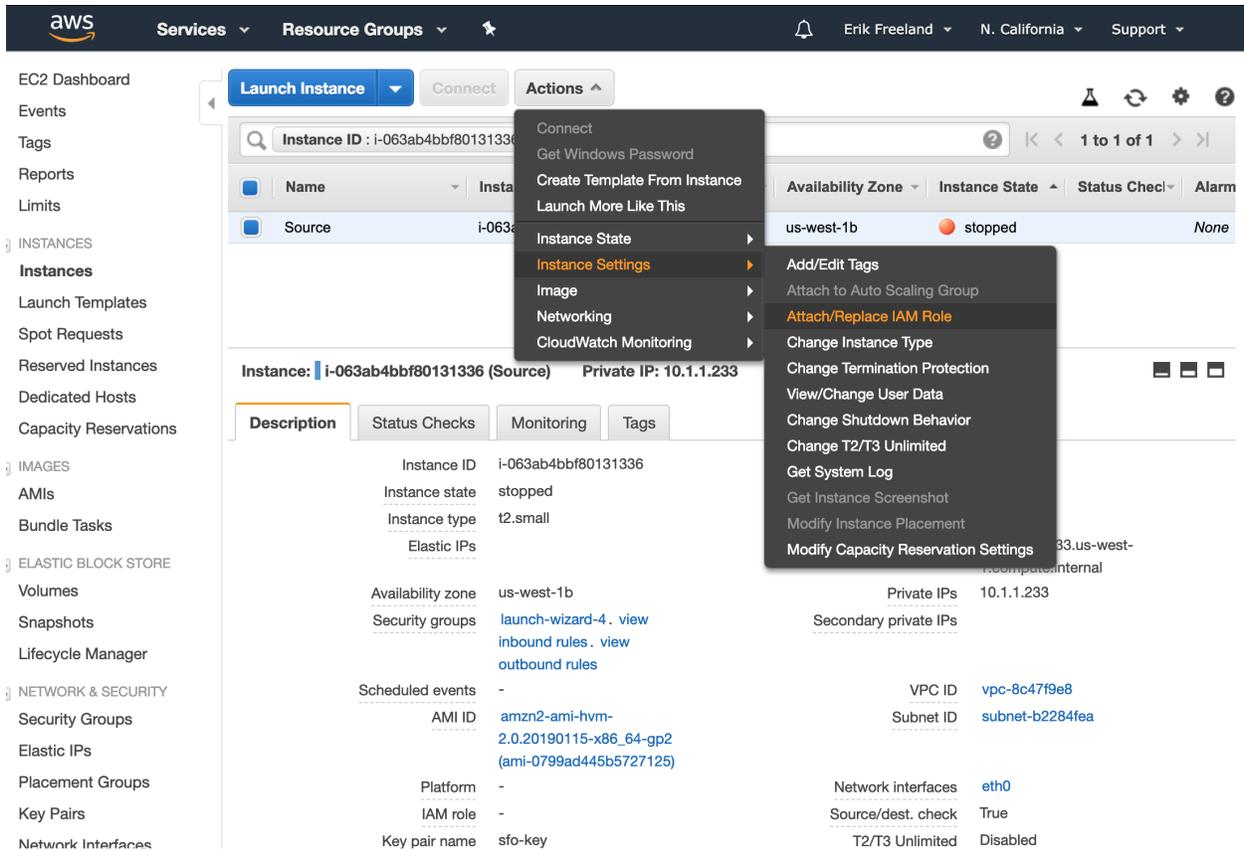
Role description

Maximum 1000 characters. Use alphanumeric and '+=, @-_' characters.

Trusted entities AWS service: ec2.amazonaws.com

Policies Policies not attached

10. The final step is to associate this IAM role with an EC2 instance. You can do this any number of ways via the CLI or automation tools. In the GUI, go back to the EC2 console and select the EC2 instance that needs this new role. Then select, Actions - Instance Settings - Attach/Replace IAM Role

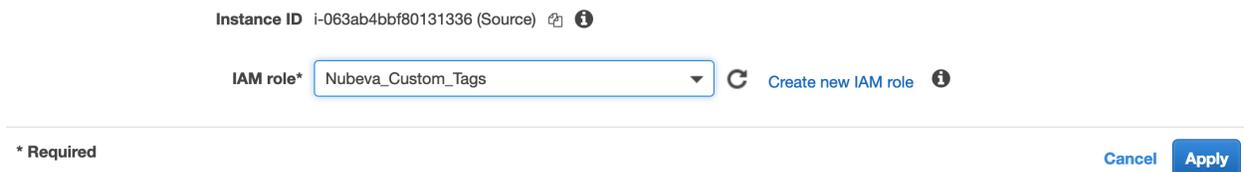


11. Choose your newly created role and hit apply. You should receive a green success message.

[Instances](#) > [Attach/Replace IAM Role](#)

Attach/Replace IAM Role

Select an IAM role to attach to your instance. If you don't have any IAM roles, choose [Create new IAM role](#) to create a role in the IAM console. If an IAM role is already attached to your instance, the IAM role you choose will replace the existing role.



12. Now, this IAM role will be attached to the EC2 instance and will be visible on the EC2 console details for this host.