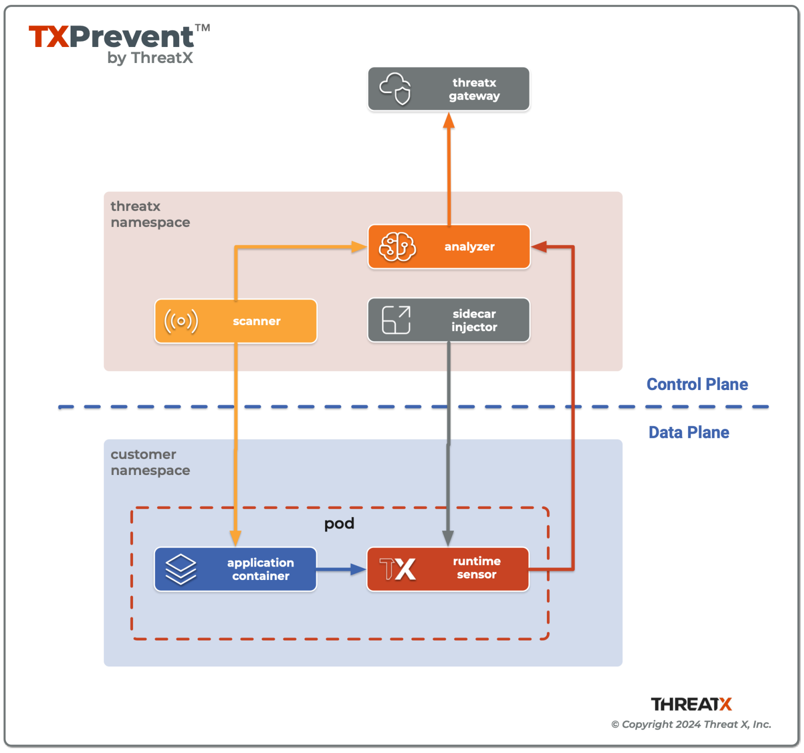
TX Prevent

Installing Prevent using Helm

Version 1.2, 2025-01-10

Introduction

This document will guide you through an installation of TX Prevent into your Kubernetes environment.



□□ Helm chart

ThreatX maintains a Helm chart to provide the best installation experience. If you are not familiar with Helm, take a moment to familiarize yourself with the Helm documentation.

Prerequisites

- Kubernetes version `>=1.27.0-0`
- Sensor API Keys
- Kubectl CLI
- Helm CLI

Example 1. Check Kubernetes Environment

```
$ kubectl version
```

Example Output

```
Client Version: v1.30.1
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.29.4-eks-036c24b
```

Install TX Prevent

A helm chart named `threatx-prevent` installs the ThreatX *Control Plane Services* and *Sensor Sidecar Injector* into the `threatx` namespace of the Kubernetes cluster.

Installing the Helm Chart

```
$ helm upgrade --namespace threatx \
    --create-namespace --install --debug \
    --set apiKey=<SENSOR_KEY> \ ①
    --set customer=<TENANT> \ ②
    --set analyzer.tags=<CLUSTER_TAGS> \ ③
    --set certManager.enabled=true \  ④
    threatx-prevent oci://public.ecr.aws/threatx/helm/threatx-prevent
```

① The `<SENSOR_KEY>` authenticates the product's component connections with ThreatX Gateway. It should not to be confused with a user's key to the ThreatX API. See Generating & revoking sensor API keys.

② The `<TENANT>` is your ThreatX tenant (customer) name.

③ Tag values for the analyzer instances. See Analyzer Tags

④ The TX Prevent services **requires TLS.** Use Cert Manager (`true`) or Helm Long-Term Self-Signed Certificate Provisioning (`false`).

Helm Tips

- Use the `--debug` switch to see all the Kubernetes configuration being applied by the chart.
- Use the `--dry-run` switch to validate the helm install without actually applying the changes.

Using a Values File

Once you know the values you want to use, you can create a `values.yml` file with the values and use the `-f` switch to install the chart (rather than `--set`).

values.yml

```
apiKey: <SENSOR_KEY>
customer: <TENANT>
analyzer:
  tags: <CLUSTER_TAGS>
certManager:
  enabled: true
```

This will be sufficient for most installations. Additional configuration options can be found in the Full Helm Configuration Reference. Change at your own risk or contact ThreatX support for assistance.

Uninstall TX Prevent

The commands in this section demonstrate complete removal of the TX Prevent control plane and sensors from your Kubernetes cluster

Remove the control plane

```
$ helm -n threatx uninstall threatx-prevent
```

Remove namespace

```
$ kubectl delete namespace threatx
```

Sensor containers will not be removed until the application pods are restarted.

Restart application pods to remove ThreatX sensors

```
$ kubectl -n my-namespace rollout restart deployment/my-application
```

Upgrading TX Prevent

Use `helm upgrade` to upgrade your version of TX Prevent.

Sensor Upgrades

Since sensors run as containers in your application pods you will need to restart those pods to get a new sensor version injected into them.

Manually restart a application deployment

```
$ kubectl -n my-namespace rollout restart deployment/my-application
```

The ThreatX Prevent Helm chart can also be configured to do rolling restarts of your application deployments during the upgrade. A set of namespaces and deployments within those namespace can be specified in the `podRestart` properties and the Helm upgrade command will perform rolling restarts of those using Helm post-upgrade hooks.

values.yml

```
podRestart:
  enabled: true
  items:
    <app-namespace>:
      deployments:
        - <app-deployment-name>
    <another-app-namespace>:
      deployments:
        - <app-deployment-name>
        - <another-app-deployment-name>
```

Configuration

This section will help you setup the *Control Plane Services*, enable *Sensor Sidecar Injector*, provision TLS certificates and define the application name.

Certificates

Communication between the ThreatX Prevent components use TLS for security. This requires the use of certificates. We provide several different options around certificate management.

Using the cert-manager Component

If cert-manager is installed in your cluster you can enable the install to use it for certificate provisioning and management.

```
certManager.enabled: true
```

Self-Signed Certificates

You can choose to have the Helm chart create self-signed certificates on installation.

```
certManager.enabled: false
```

Certificate Renewal

The self-signed certificates created on install are good for 2 years. To renew the self-signed certificates perform a `helm upgrade` with a configuration property of `renewCerts=true`. After the upgrade command runs you will need to restart the control plane services:

```
$ kubectl -n threatx rollout restart deployment/threatx-analyzer
$ kubectl -n threatx rollout restart deployment/threatx-sts
```

All application pods with sensors will also need to be restarted (See Upgrading TX Prevent)

External Secrets

You can also choose to manage the product certificate secrets outside of the Helm chart, you must use these Kubernetes secret names and set the `externalSecret` property to `true`.

Certificate Authority (CA) Names	`threatx-analyzer-ca-tls` or `threatx-sts-ca-tls`
TLS Secret (certificate) Names	`threatx-analyzer-server-tls` or `threatx-sts-server-tls`

values.yml

```
externalSecrets:
  enabled: true
```

Self Managed Certificates

If you want to self provision the product certificates and then pass them into the installation you can use the following properties.

These values must be provided as **base64** encoded strings.

values.yml

```
# For self-managed Analyzer certificates
analyzer:
  caCert:
  serverCert:
  serverfKey:
# For self-managed STS certificates
sts:
  caCert:
  serverCert:
  serverfKey:
```

Analyzer & Scanning Template Service (STS)

□□ Application Name

For the most accurate tracking of events at the application level the ThreatX Protect sensor needs to derive the name of the application that it is monitoring in the pod. This is done by looking at the pod labels.

The `applicationNameLabels` property in the Helm chart is a list of pod label names that are used to derive the application name. The default list is:

- `app.kubernetes.io/name`
- `app`
- `name`

If your application uses a different pod label for the application name, you can add it to the list as a helm configuration property.

□□ Analyzer Tags

The tagging of analyzer instances is done with the `analyzer.tags` property. The value of this should be a comma-separated list of strings that can identify the set of analyzers in a particular deployment.

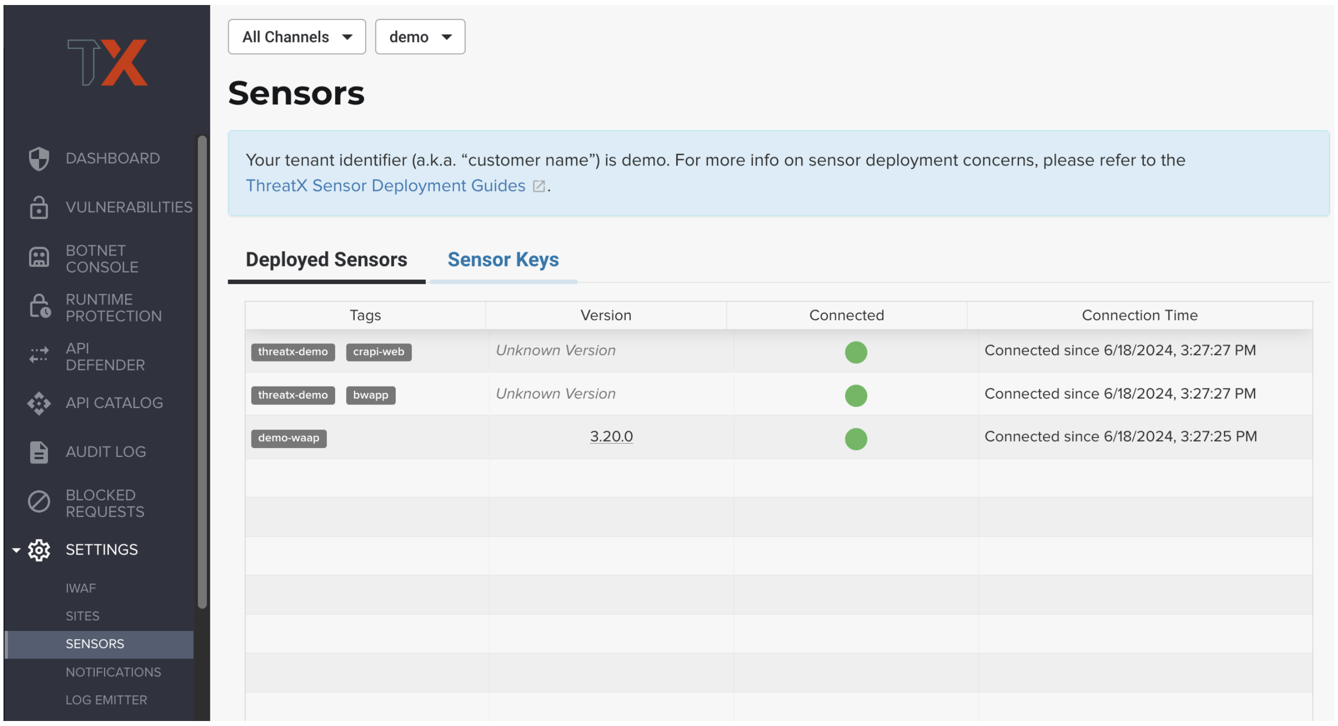
Single tag

```
analyzer.tags=production
```

Multiple tags

```
analyzer.tags=production,pci,east
```

These tags are visible in the CtrlX Sensor Dashboard



Sensors

Your tenant identifier (a.k.a. “customer name”) is demo. For more info on sensor deployment concerns, please refer to the ThreatX Sensor Deployment Guides.

Tags	Version	Connected	Connection Time
threatx-demo crapi-web	*Unknown Version*	🟢	Connected since 6/18/2024, 3:27:27 PM
threatx-demo bwapp	*Unknown Version*	🟢	Connected since 6/18/2024, 3:27:27 PM
demo-waap	3.20.0	🟢	Connected since 6/18/2024, 3:27:25 PM

Figure 1. Analyzer tags seen as Tags on the ThreatX Sensors page.

Each of the *Deployed Sensors* represents a single instance of an **Analyzer**, which in turn can have multiple connected sensors.

Analyzer event sampling

The `analyzer.enableSampling` property controls the sampling of API Analyzer events.

When enabled, it caches duplicate API Analyzer Events to reduce the number reported to the ThreatX backend. It is enabled by default to reduce egress traffic

We recommend setting the sampling to `false` when initially testing out a deployment, but then flipping it back to `true` after the deployment has been verified.

Runtime Sensor Deployment

Sidecar Injector

The *Sidecar Injector* is a Kubernetes Mutating Admission Webhook service that will inject ThreatX runtime sensor containers into application pods based upon the presence of a pod label.

Adding a sensor to an application

The Sidecar Injector will inject the runtime sensor container into any pods created with this label

```
inject-threatx-sidecar: "true"
```

Sample Pod resource spec with inject label added

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: sample-app
    inject-threatx-sidecar: "true"
  name: sample-app
  namespace: sample
spec:
  containers:
  - name: sample-app
 # ...
```

This label should typically be added to the application’s Kubernetes Deployment, Statefulset, or Daemonset resource. This will ensure that all created pods by that resource get a sensor injected.

Sample Deployment resource spec with inject label added

```yaml
apiVersion: apps/v1
```

```yaml
kind: Deployment
metadata:
  name: sample-app
  namespace: sample
spec:
  progressDeadlineSeconds: 600
  replicas: 2
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
        inject-threatx-sidecar: "true"
    spec:
      containers:
      - name: dvwa
    # ...
```

Simply adding the label to a resource with pre-existing pods will not automatically inject those pods; **you will need to restart them** (e.g. with `kubectl rollout restart` and so on). This is because Kubernetes does not call the webhook until it needs to start the underlying resources.

Sensor injection can be disabled at the namespace level with the following label

```
config.threatx.io/admission-webhooks: disabled
```

Sidecar injection is always disabled for the `kube-system` namespace.

□ Verifying a sensor is running

The sensor can be verified after injection by describing the application pod. The `public.ecr.aws/threatx/threatx-runtime-sensor` image should be seen running inside the pod.

□□ Helm Chart Configuration Reference

Table 1. All Helm Chart Values

Key	Type	Default	Description
`apiKey`	string	""	Your ThreatX api key
`customer`	string	""	Your ThreatX customer ID

Key	Type	Default	Description
`certManager.enabled`	boolean	false	Use your cluster's cert-manager component to provision certificates for the sidecar injector. See Certificates
`analyzer.enabled`	boolean	true	Install the Runtime Analyzer service
`analyzer.instances`	int	2	The number of Analyzer instances to run
`analyzer.image.repository`	string	public.ecr.aws/threatx/threatx-runtime-analyzer"	Runtime Analyzer image repository
`analyzer.image.tag`	string	1.2.0	Runtime Analyzer image tag
`analyzer.image.pullPolicy`	string	"IfNotPresent"	Runtime Analyzer image pull policy. See Image Pull Policy for more information.
`analyzer.gatewayHostname`	string	threatx-grpc2kafka-production-v1.xplat-production.threatx.io	The hostname of the ThreatX gateway server
`analyzer.tags`	string	""	Tags for your ThreatX analyzers which are visible in the CtrlX dashboard
`analyzer.tlsEnabled`	boolean	true	TLS enabled for sensor to analyzer communication
`analyzer.externalSecret`	boolean	false	The secrets for the analyzer will be managed outside of the Helm chart. See External Secrets
`analyzer.caCert`	string	""	The base64 encoded CA pem to use for the Analyzer. See Self Managed Certificates
`analyzer.serverCert`	string	""	The base64 encoded CA pem to use for the Analyzer. See Self Managed Certificates
`analyzer.serverKey`	string	""	The base64 encoded CA pem to use for the Analyzer. See Self Managed Certificates
`analyzer.stsClientSink`	string	"NoneStsClient"	ThreatX STS service output target
`analyzer.rawAaeSendCompressed`	boolean	false	compress the API Analyzer Events sent from the Analyzer to STS
`analyzer.rawAaeAcceptCompressed`	boolean	false	allow compressed events from STS
`analyzer.enableSampling`	boolean	true	cache duplicate API Analyzer Events to reduce the number sent to the ThreatX backend
`analyzer.stsClientSink`	string	"ApiAnalyzerEventClient"	Client sink name
`analyzer.stsPort`	int	443	The port number of the STS service
`analyzer.stsTlsEnabled`	boolean	true	Enable TLS with the STS service

Key	Type	Default	Description
`analyzer.logLevel`	string	"info"	The logging level
`analyzer.backtrace`	int	1	The logging backtrace level
`analyzer.resources.requests.cpu`	string	"500m"	Amount of CPU units that the Runtime Analyzer container requests for scheduling. See Requests and Limits for more information.
`analyzer.resources.requests.memory`	string	"500Mi"	Amount of memory that the Runtime Analyzer container requests for scheduling. See Requests and Limits for more information.
`analyzer.resources.limits.cpu`	string	"2"	Maximum amount of CPU units that the Runtime Analyzer container can use. See Requests and Limits for more information.
`analyzer.resources.limits.memory`	string	"2G"	Maximum amount of memory that the Runtime Analyzer container can use. Requests and Limits for more information.
`analyzer.scaling.enabled`	boolean	true	Create a horizontalpodautoscaler for the Runtime Analyzer service
`analyzer.scaling.minReplicas`	int	2	The minimum number of Runtime Analyzer instances to run
`analyzer.scaling.maxReplicas`	int	6	The maximum number of Runtime Analyzer instances to run
`analyzer.scaling.cpuUtilPercentage`	int	200	The percentage of the request cpu limit (analyzer.resources.requests.cpu) to use as a scaling threshold. See: How does a horizontalpodautoscaler work
`otel.enabled`	boolean	true	Install the Threatx OTEL service
`otel.hostname`	string	""	The hostname of the ThreatX OTEL server that will receive log and event data
`otel.*`			Options for the Grafana Alloy Helm Chart, See: grafana.com/docs/alloy/latest/configure/kubernetes
`sensor.image.repository`	string	public.ecr.aws/threatx/threatx-runtime-sensor"	ThreatX Prevent sensor image repository
`sensor.image.tag`	string	1.2.0	ThreatX Prevent sensor image tag
`sensor.image.pullPolicy`	string	"IfNotPresent"	ThreatX Prevent sensor image pull policy. See Image Pull Policy for more information.

Key	Type	Default	Description
`sensor.applicationNameLabels`	list	["app.kubernetes.io/name","app","name"]	List of pod labels to use for deriving the pod's application name. See Sensor Tags
`sensor.interfaceName`	string	"eth0"	The host network interface name.
`sensor.tracingPath`	string	"/sys"	The host tracing path. S
`sensor.logLevel`	string	"info"	The logging level
`sensor.backtrace`	int	1	The logging backtrace level
`sensor.targetEnvironment`	string	"k8s-sidecar"	The target environment that the sensor will be running in
`sensor.resources.requests.cpu`	string	"100m"	Amount of CPU units that the ThreatX Prevent sensor container requests for scheduling. See Requests and Limits for more information.
`sensor.resources.requests.memory`	string	"250Mi"	Amount of memory that the ThreatX Prevent sensor container requests for scheduling. See Requests and Limits for more information.
`sensor.resources.limits.cpu`	string	"250m"	Maximum amount of CPU units that the ThreatX Prevent sensor container can use. See Requests and Limits for more information.
`sensor.resources.limits.memory`	string	"250Mi"	Maximum amount of memory that the ThreatX Prevent sensor container can use. See Requests and Limits for more information.
`sts.enabled`	boolean	true	Install the ThreatX Prevent Scan Template Service (STS)
`sts.instances`	int	2	The number of Scan Template Service instances to run
`sts.image.repository`	string	public.ecr.aws/threatx/threatx-sts"	Scan Template Service image repository
`sts.image.tag`	string	1.1.0	Scan Template Service image tag
`sts.image.pullPolicy`	string	"IfNotPresent"	Scan Template Service image pull policy. See Image Pull Policy for more information.
`sts.grpcTlsEnabled`	boolean	true	TLS enabled
`sts.grpcListenPort`	string	"50051"	The gRPC listener port
`sts.externalSecret`	boolean	false	The secrets for the STS will be managed outside of the Helm chart. See External Secrets
`sts.caCert`	string	""	The base64 encoded CA pem to use for the STS. See Self Managed Certificates
`sts.serverCert`	string	""	The base64 encoded CA pem to use for the STS. See Self Managed Certificates

Key	Type	Default	Description
`sts.serverKey`	string	""	The base64 encoded CA pem to use for the STS. See Self Managed Certificates
`sts.logLevel`	string	"info"	The logging level
`sts.resources.requests.cpu`	string	"500m"	Amount of CPU units that the STS container requests for scheduling. See Requests and Limits for more information.
`sts.resources.requests.memory`	string	"500Mi"	Amount of memory that the STS container requests for scheduling. See Requests and Limits for more information.
`sts.resources.limits.cpu`	string	"2"	Maximum amount of CPU units that the STS container can use. See Requests and Limits for more information.
`sts.resources.limits.memory`	string	2G"	Maximum amount of memory that the STS container can use. See Requests and Limits for more information.
`sts.scaling.enabled`	boolean	true	Create a horizontalpodautoscaler for the STS service
`sts.scaling.minReplicas`	int	2	The minimum number of STS instances to run
`sts.scaling.maxReplicas`	int	6	The maximum number of STS instances to run
`sts.scaling.cpuUtilPercentage`	int	200	The percentage of the request cpu limit (sts.resources.requests.cpu) to use as a scaling threshold. See: kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/#how-does-a-horizontalpodautoscaler-work
`sidecarInjector.enabled`	boolean	true	Install the ThreatX Prevent Sidecar Injector service
`sidecarInjector.image.repository`	string	public.ecr.aws/threatx/threatx-sidecar-injector"	ThreatX Prevent sidecar injector image repository
`sidecarInjector.image.tag`	string	1.1.0	ThreatX Prevent sidecar injector image tag
`sidecarInjector.image.pullPolicy`	string	"IfNotPresent"	ThreatX Prevent sidecar injector image pull policy. See Image Pull Policy for more information.
`sidecarInjector.resources.requests.cpu`	string	"100m"	Amount of CPU units that the ThreatX Prevent sidecar injector container requests for scheduling. See Requests and Limits for more information.

Key	Type	Default	Description
`sidecarInjector.resources.requests.memory`	string	"100Mi"	Amount of memory that the ThreatX Prevent sidecar injector container requests for scheduling. See Requests and Limits for more information.
`sidecarInjector.resources.limits.cpu`	string	"200m"	Maximum amount of CPU units that the ThreatX Prevent sidecar injector container can use. See Requests and Limits for more information.
`sidecarInjector.resources.limits.memory`	string	"200Mi"	Maximum amount of memory that the ThreatX Prevent sidecar injector container can use. See Requests and Limits for more information.
`renewCerts`	boolean	false	Regenerate certificates for the control plane services.