AppSentinels.ai
Application Security Re-invented

AppSentinels API Security Platform

Front Proxy Deployment

Contents

Revision	Date Modified	Author	Comments
1.0	02-Jan-24	Sachin	Initial Draft
1.1	24-Oct-24	Sachin	Further explanations for TLS and mTLS
1.2	11-Oct-24	Sachin	Side car proxy support, ECS examples
1.3	04-Mar-25	Sachin	mTLS for K8 sidecar

1. Introduction

AppSentinels provides a front proxy to integrate with your existing deployments. This deployment serves cases of

1. Monolith application/service requiring AppSentinels inspection
2. End to end encrypted systems requiring a point of inspection

The front proxy has the capability to terminate SSL traffic and re-encrypt the upstream traffic.

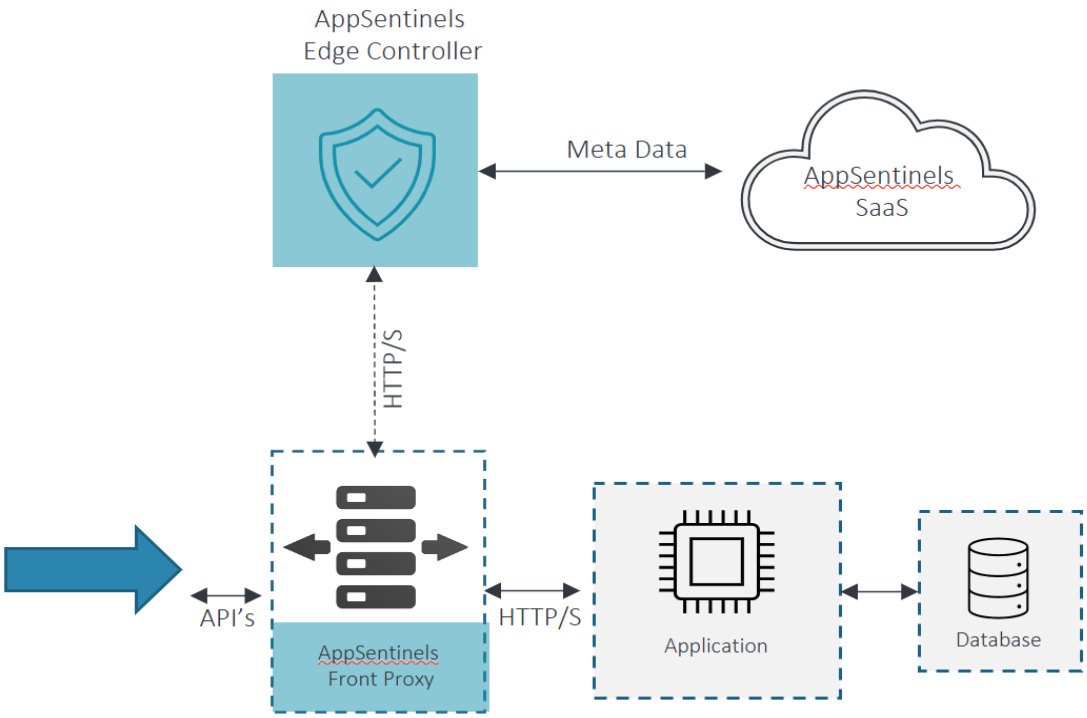

Figure 1 Front proxy deployment with AppSentinels

1. AppSentinels front proxy gets HTTP/HTTPS traffic and forward the logs to AppSentinels Edge Controller for security processing.
2. Modules support two modes configurable via a knob Out-of-Band(OOB)/Transparent & Service-chaining/Enforcement. In both modes, AppSentinels process a copy of the packet.
3. In OOB/transparent mode, plugin forwards the packet to the Application and Edge controller simultaneously. In service-chaining mode, the plugin forwards the packet to Edge Controller and waits for its output before forwarding the packet to Application. This allows the plugin to enforce inline action based on response received from Edge Controller.
4. AppSentinels Service-chaining mode has optional max-latency configuration. In case Edge controller response is delayed and latency crosses configured threshold, plugin gets into fail-open mode and forwards the packet to Application thereby ensuring availability and responsiveness in case of a slowness or an outage.

2. Deployment Options

Front proxy can be deployed as a container service.

2.1 Deployment using dockers

```
services:
  front-service:
    image: appsentinels/front-proxy:latest
    env_file:
      - ./env/env-envoy.txt
    expose:
      - ${ENVOY_LISTENING_PORT}
    ports:
      - "${ENVOY_EXTERNAL_DATA_LISTENING_PORT}:${ENVOY_LISTENING_PORT}"
    volumes:
      - /var/crash:/var/crash
```

a. The service can be tuned by environment variables that can be passed via env_file specification
b. By default, the service assumes it will be listening on port 8000, defined by,
    a. ENVOY_EXTERNAL_DATA_LISTENING_PORT and
    b. ENVOY_LISTENING_PORT

2.1.1 Configuration via environmental variables

The below configurable parameters provide the necessary changes needed for any deployment.

Environmental Config	Description	Default
ENVOY_SERVICE_NAME	Upstream service name or hostname or IP address this proxy is front ending	-
ENVOY_SERVICE_PORT	Upstream service listening port	-
ENVOY_INSTANCE_NAME	Visibility information of this proxy. Provide a meaningful name for this instance based on deployment	unknown
ENVOY_ON_PREM_CONTROLLER	Edge controller's service name or hostname or IP addresses	-
ENVOY_DOWNSTREAM_TLS_SERVER_CERT	Mounted cert location needed for accepting downstream TLS requests. If defined along with ENVOY_DOWNSTREAM_TLS_SERVER_PRIVATE_KEY, proxy will start accepting TLS client requests	-
ENVOY_DOWNSTREAM_TLS_SERVER_PRIVATE_KEY	Private key for downstream TLS requests	-
ENVOY_DOWNSTREAM_TLS_CA_CERT	For downstream mTLS, provide trust cert for client cert validation. Client cert is validated only if this is defined	-
ENVOY_UPSTREAM_TLS	Setting to true will enable upstream TLS with no server validation	false
ENVOY_UPSTREAM_TLS_CLIENT_CERT	For upstream mTLS, location where client cert presented by front proxy is present. This along with ENVOY_UPSTREAM_TLS_CLIENT_PRIVATE_KEY will enable upstream TLS	-
ENVOY_UPSTREAM_TLS_CLIENT_PRIVATE_KEY	For upstream mTLS, provide private key	-
ENVOY_UPSTREAM_TLS_CA_CERT	For upstream mTLS, provide trust cert for upstream server validation if ENVOY_UPSTREAM_TLS=true	-
PROXY_SIDECAR	Set to yes for using proxy as a sidecar with no changes to application's external listening port.	no

	Please ensure, ENVOY_LISTENING_PORT is set to any available port if this option is set to yes	
ENVOY_LISTENING_PORT	Any available port for envoy to listen on in case of proxy sidecar deployment. This is internal to envoy.	-

Please note that for certificates will need to be mounted at the locations specified via the above environments.

2.1.2 Starting front proxy service

Provided the above environment variables have been defined in a file called env-envoy.txt, the following commands can be used to start the front proxy service,

```
docker-compose -f docker-compose.yaml --env=env/env-envoy.txt up -d
```

In some versions of docker compose, it will be required to populate environment variables before starting the service, in which case, please use,

```
export ENVOY_EXTERNAL_DATA_LISTENING_PORT=8000;\
export ENVOY_LISTENING_PORT=8000;\
docker-compose -F docker-compose.yaml --env=env/env-envoy.txt up -d
```

Reference configuration can be found at https://sample-config.appsentinels.ai/appsentinels-deployment/sample-docker-compose/docker-compose-monolith.yaml (refer to the front-service)

2.2 Upstream MTLS deployment

Apart from setting the correct environment variables, as below, the certificates and keys will have to be mounted.

```
# if provided, envoy will start a TLS server downstream, otherwise plain text
ENVOY_DOWNSTREAM_TLS_SERVER_CERT=/etc/server_public.crt
ENVOY_DOWNSTREAM_TLS_SERVER_PRIVATE_KEY=/etc/server_private.key

#upstream tls true doesnt mean mTLS, for mTLS we need to provide cert and key
ENVOY_UPSTREAM_TLS=true

# server validation is performed if this is provided, by default no server validation is done
ENVOY_UPSTREAM_TLS_CA_CERT=/etc/ssl/certs/ca-certificates.crt

# client cert and key for upstream TLS
ENVOY_UPSTREAM_TLS_CLIENT_CERT=/etc/client_cert.crt
ENVOY_UPSTREAM_TLS_CLIENT_PRIVATE_KEY=/etc/client_private.key
```

Mounting of the certificates can be done using docker-compose as below,

```
services:
  front-service:
    image: appsentinels/front-proxy:latest
    env_file:
      - ./env/env-envoy.txt
```

```
expose:
  - ${ENVOY_LISTENING_PORT}
ports:
  - "${ENVOY_EXTERNAL_DATA_LISTENING_PORT}:${ENVOY_LISTENING_PORT}"
volumes:
  - /var/crash:/var/crash
  - ./server_public.pem:/etc/server_public.crt # downstream TLS cert
  - ./server_private.pem:/etc/server_private.key # downstream TLS key
  - ./server_public.pem:/etc/client_cert.crt # client cert to server
  - ./server_private.pem:/etc/client_private.key # client key
  - ./externalenvoy.yaml:/tmp/envoy.yaml
```

In the above example, front proxy uses the same set of keys and certs for upstream and downstream. Please do change this based on deployment.

For server validation, CA cert (pointed by ENVOY_UPSTREAM_TLS_CA_CERT) can be system cert or a custom like. In case of custom, the same should be mounted like the other certs.

2.3 Downstream MTLS deployment

If is it desired that client cert be validated, you can define, ENVOY_DOWNSTREAM_TLS_CA_CERT to point to trust cert to validate. This cert can be mounted similar to other certs or just point to a system CA cert.

2.4 Secure Logging

Front proxy can perform logging over HTTPS. This can be achieved by simply adding the below environment,

```
ENVOY_AUTHZ_SERVICE=ext_authz-https-service
```

2.5 Resource Requirements

Front proxy throughput will be defined by the resources it will allocated. Below table acts as reference for estimating how much should be CPU and memory should be allocated for a given throughput.

CPU	Memory	Throughput (API request per second)
0.5	256M	250 req/sec
1	512M	400 req/sec
2	2G	1000 req/sec

2.6 Kill Switch

Front proxy performs health check on the logging endpoint (usually an edge controller or an intermediate load balancer). If the health of the logging endpoint is detected to be bad, front proxy will cease logging to conserve the resources, although the health checks will continue to be performed. The logging endpoint is assumed to be bad if it returns a non-200 status code.
As part of policy push, edge controllers will be configured the desired logging mode for the sensors (like front proxy). If logging mode is no-logging or in kill switch mode, edge controller will respond with non-200 to every logging or health check from the front proxy.

The behavior with an intermediate load balancer will remain the same. The load balancer is expected to perform health checks on the upstream edge controllers and respond back accordingly to the front proxy.

2.7 Advanced configurations

Front proxy is currently built on top of envoy proxy. While the previous method provides for a much simpler and easier deployment, the front proxy allows for a direct configuration via envoy yaml. To do, so one can define their own envoy.yaml and mount it inside the front proxy image and define the below environment variable,

```
ENVOY_USE_THIS_CONFIG_FILE=/tmp/externalenvoy.yaml
```

```yaml
services:
  front-service:
    image: appsentinels/front-proxy:latest
    env_file:
      - ./env/env-envoy.txt
    expose:
      - ${ENVOY_LISTENING_PORT}
    ports:
      - "${ENVOY_EXTERNAL_DATA_LISTENING_PORT}:${ENVOY_LISTENING_PORT}"
    volumes:
      - /var/crash:/var/crash
      - ./externalenvoy.yaml:/tmp/externalenvoy.yaml
```

2.8 ECS Support

2.8.1 ECS on EC2

Front proxy can be deployed in side car mode for ECS on EC2 without having to change the application ports.

Please note

1. Capability "NET_ADMIN" needs to be added for side car proxy mods
2. Sidecar mode isnt supported on older kernels (~4.1x) of Amazon Linux 2

Below sample config provides an example of an application listening on HTTPS port 8091 and front proxy sidecar deployed along with it. This example also includes secure logging onto controller.

```json
{
    "taskDefinitionArn": "....",
    "containerDefinitions": [
        {
            "name": "loopback",
            "image": "appsentinels/http-server:latest",
            "portMappings": [
                {
                    "name": "loopback-8091-tcp",
                    "containerPort": 8091,
                    "hostPort": 8091,
                    "protocol": "tcp",
                    "appProtocol": "http"
                }
            ],
```

```
        },
        {
            "name": "front-proxy",
            "image": "appsentinels/front-proxy:latest",
            "portMappings": [
                {
                    "name": "proxy",
                    "containerPort": 8000,
                    "hostPort": 8000,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "environment": [
                {
                    "name": "ENVOY_UPSTREAM_TLS",
                    "value": "true"
                },
                {
                    "name": "ENVOY_DOWNSTREAM_TLS_SERVER_CERT",
                    "value": "/etc/server_public.crt"
                },
                {
                    "name": "ENVOY_DOWNSTREAM_TLS_SERVER_PRIVATE_KEY",
                    "value": "/etc/server_private.key"
                },
                {
                    "name": "ENVOY_SERVICE_NAME",
                    "value": "127.0.0.1"
                },
                {
                    "name": "PROXY_SIDECAR",
                    "value": "yes"
                },
                {
                    "name": "ENVOY_LISTENING_PORT",
                    "value": "8000"
                },
                {
                    "name": "ENVOY_SERVICE_PORT",
                    "value": "8091"
                },
                {
                    "name": "ENVOY_ON_PREM_CONTROLLER",
                    "value": "10.101.3.242"
                },
                {
                    "name": "ENVOY_AUTHZ_SERVICE",
                    "value": "ext_authz-https-service"
                }
            ],
            "linuxParameters": {
                "capabilities": {
                    "add": [
                        "NET_ADMIN"
                    ],
                    "drop": []
                }
            },
        }
    ],
    "family": "ECS-ON-EC2-HTTP-14-11-2024",
    "executionRoleArn": "arn:aws:iam::488922454646:role/ecsTaskExecutionRole",
    "networkMode": "awsvpc"
}
```

2.8.2 Fargate

Fargate is very restrictive in-terms of capabilities provided for sensor deployment. Fargate doesn't allow for NET_ADMIN or any other privilege escalation. Hence front proxy cannot be deployed in side car proxy mode. The only solution is for front proxy to own the existing application ports and changing of the original application ports.

Below is an example of where earlier application '**loopback'** was listening on port 443 but on injection of sidecar, will listen on 8443. Port 443 being now owned by envoy defined by ENVOY_LISTENING_PORT

```
{
    "containerDefinitions": [
        {
            "name": "loopback",
            "image": "appsentinels/http-server:latest",
            "cpu": 0,
            "portMappings": [
                {
                    "name": "loopback-8443-tcp",
                    "containerPort": 8443,
                    "hostPort": 8443,
                    "protocol": "tcp",
                    "appProtocol": "http"
                }
            ],
            "environment": [
                {
                    "name": "HTTPS_SERVICE_PORT",
                    "value": "8443"
                }
            ]
        },
        {
            "name": "sidecar",
            "image": "appsentinels/front-proxy:latest",
            "cpu": 0,
            "portMappings": [
                {
                    "name": "sidecar-443-tcp",
                    "containerPort": 443,
                    "hostPort": 443,
                    "protocol": "tcp",
                    "appProtocol": "http"
                }
            ],
            "essential": true,
            "environment": [
                {
                    "name": "ENVOY_UPSTREAM_TLS",
                    "value": "true"
                },
                {
                    "name": "ENVOY_DOWNSTREAM_TLS_SERVER_CERT",
                    "value": "/etc/server_public.crt"
                },
                {
                    "name": "ENVOY_DOWNSTREAM_TLS_SERVER_PRIVATE_KEY",
                    "value": "/etc/server_private.key"
                },
                {
                    "name": "ENVOY_SERVICE_NAME",
```

```
                "value": "127.0.0.1"
            },
            {
                "name": "ENVOY_LISTENING_PORT",
                "value": "443"
            },
            {
                "name": "ENVOY_SERVICE_PORT",
                "value": "8443"
            },
            {
                "name": "ENVOY_ON_PREM_CONTROLLER",
                "value": "10.101.3.242"
            },
            {
                "name": "ENVOY_AUTHZ_SERVICE",
                "value": "ext_authz-https-service"
            }
        ]
    }
  ]
}
```

2.9 K8 Sidecar Support

Front proxy can be inserted as a sidecar in existing K8 deployments. Front proxy can terminate TLS connections on behalf of application and also perform mutual TLS. Please refer to section Configuration via environmental variables.

Front proxy will need to be passed environmental variables as configuration. The variables usually point to certificate locations which can turn on required TLS or mTLS features.

Below is a sidecar deployment for establishing mTLS for both upstream and downstream. It unlikely that upstream will require mTLS as the communication will be within the pod.

```
kind: Deployment
...
    spec:
      volumes:
        - name: domain-server-secret
          secret:
              secretName: domain-server-secret
      containers:
        # Sample application, listens on port 9010 for HTTPS and 9000 for HTTP
        - name: http-service
          image: appsentinels/http-server:latest
          imagePullPolicy: IfNotPresent
          ports:
            - name: getextractport
              containerPort: 9010
        - name: front-proxy
          image: appsentinels/front-proxy:latest
          imagePullPolicy: IfNotPresent
          volumeMounts:
             - name: domain-server-secret
               mountPath: /etc/server_public.crt
               subPath: app1.crt
               readOnly: true
             - name: domain-server-secret
```

```
              mountPath: /etc/server_private.key
              subPath: app1.key
              readOnly: true
            # mTLS: CA cert for validating client certificate (load balancer or app cert validation)
            - name: domain-server-secret
              mountPath: /etc/ssl/certs/ca-certificates.crt
              subPath: myCA.crt
              readOnly: true
          securityContext:
             capabilities:
               # Needed for installing filters to punt traffic to the sidecar
               add:
                - NET_ADMIN
          env:
            # On-prem controller to send logs to, update with IP or hostname (no schema or port needed)
            - name: ENVOY_ON_PREM_CONTROLLER
              value: <controller-ip-or-hostname>

            # Keep it same as the target port of container
            - name: ENVOY_SERVICE_PORT
              value: "<target container port, in this case 9010>"

            # Downstream side TLS certs, this will same as the certs applications were presenting earlier
            # Server cert presented by sidecar to other apps or load balancers downstream
            - name: ENVOY_DOWNSTREAM_TLS_SERVER_CERT
              value: "/etc/server_public.crt"
            - name: ENVOY_DOWNSTREAM_TLS_SERVER_PRIVATE_KEY
              value: "/etc/server_private.key"

            # mTLS: Define this client location if sidecar needs to be validate client certificate
            - name: ENVOY_DOWNSTREAM_TLS_CA_CERT
              value: "/etc/ssl/certs/ca-certificates.crt"

            # Upstream side TLS certs
            # Define this if sidecar needs to be validate server (i.e. app) certificate. It is unlikely that
intra-pod
            # communication will be TLS, otherwise set this to "false"
            - name: ENVOY_UPSTREAM_TLS
              value: "true"
            # For upstream mTLS, location where client cert presented by front proxy to app is present. This
along with ENVOY_UPSTREAM_TLS_CLIENT_PRIVATE_KEY will enable upstream TLS
            # enable upstream TLS
            - name: ENVOY_UPSTREAM_TLS_CLIENT_CERT
              value: "/etc/server_public.crt"
            - name: ENVOY_UPSTREAM_TLS_CLIENT_PRIVATE_KEY
              value: "/etc/server_private.key"

            # CA cert for upstream server certificate validation
            - name: ENVOY_UPSTREAM_TLS_CA_CERT
              value: "/etc/ca-certificate.crt"

            # Leave this to "127.0.0.1" for sidecar in K8 pods
            - name: ENVOY_SERVICE_NAME
              value: "127.0.0.1"

            # No change needed here
            - name: PROXY_SIDECAR
              value: "yes"
```

```
        - name: ENVOY_LISTENING_PORT
          value: "8000"
```

Sample Spec: https://sample-config.appsentinels.ai/appsentinels-deployment/k8/sidecar/front-proxy-sidecar-mtls.yaml

3. Verify Deployment

The latest version of AppSentinels servers provides sensor visibility. Once the proxy is brought and verified to be running correctly (via docker ps), it should be visible on the dashboard under organization's system health page.

