AppSentinels.ai
Application Security Re-invented

AppSentinels API Security Platform

SSL Sniffer Deployment (eBPF)

Table of Contents

Revision	Date Modified	Author	Comments
1.0	17-Nov-24	Sachin Desai	Initial Draft
1.1	20-Mar-25	Sachin Desai	RHEL baremetal support
1.2	01-Apr-25	Sachin Desai	Kubernetes Daemonset
1.3	09-May-25	Sachin Desai	Ingress controller sniffing update

Introduction

AppSentinels SSLSniffer Sensor deployment is Out-Of-Band deployment mode, without any modification required in the application configuration and application environment.

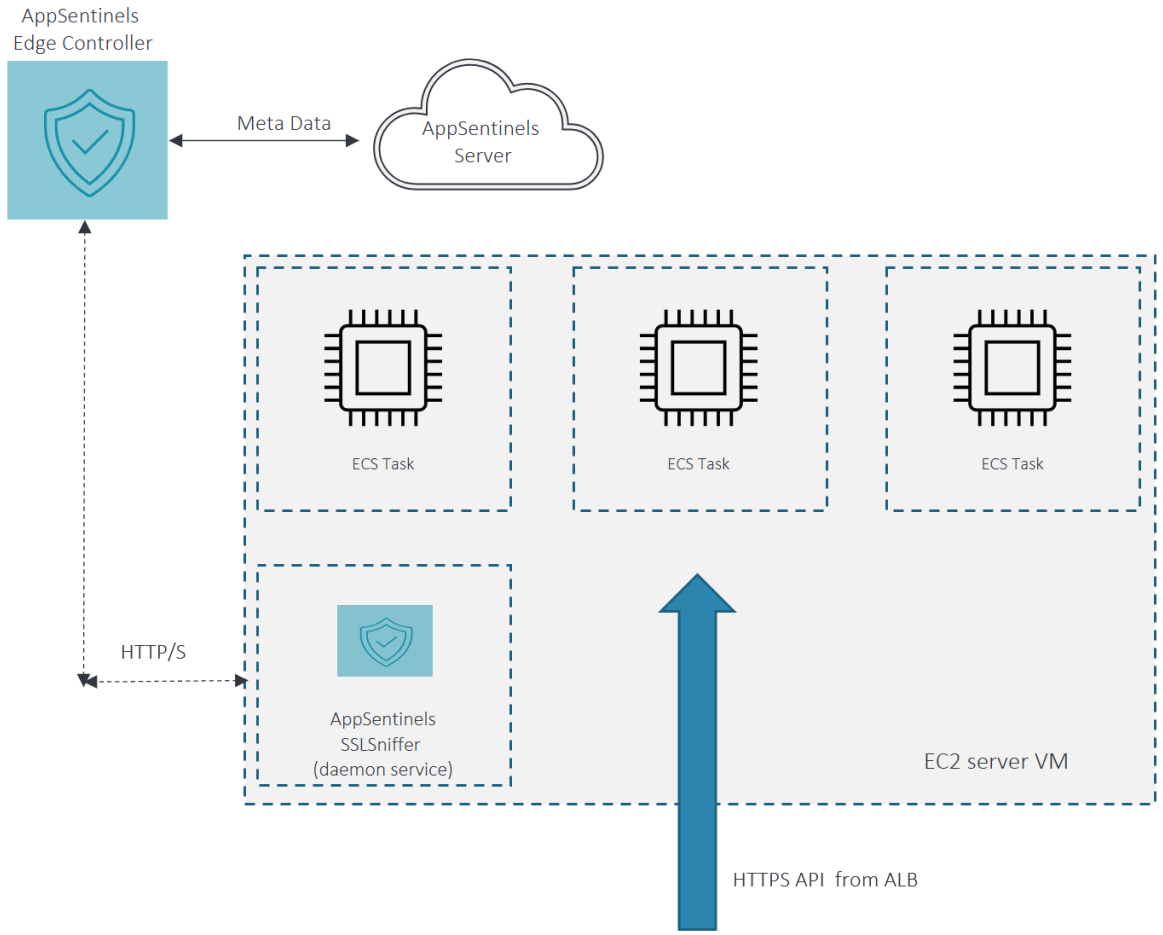
AppSentinels SSLSniffer Sensor (called sensor henceforth) can be deployed along side Application instances. This type of sensor is specially useful for deployments where there exists end to end HTTPS encryption of API traffic.

The Sniffer Sensor uses eBPF mechanism to capture the HTTPS traffic and forwards the API traffic to the AppSentinels Edge Controller (called controller henceforth).

To take preventive action against Security Events and Threat Actors requires integration of Firewall or API Gateway with AppSentinels API Security Platform since this deployment is in Out-Of-Band mode.

Deployment

ECS Deployment - Topology



SSLSniffer should be deployed alongside ECS containers in a daemon mode (single instance per EC2 VM). The container will monitor processes running in containers (or select containers) and record plain text API request and responses before they are encrypted for SSL.

Once the API log is formed at the SSLSniffer, it will be logged over to the edge controller over HTTP or HTTPS.

Configurations

Environment	Default	Description
REMOTE_CONTROLLER_SERVER_ENDPOINT	-	Edge controller endpoint with scheme, host and path. Eg: https://edge-controller:9004/mergedlog
INSTANCE_NAME	Unknown	Instance name for visibility. Useful to identify sensors on the dashboard, mention something meaningful here
SSLSNIFF_CONTAINER_INCLUDE_FILTER	-	By default all the containers are monitored, but specify only particular container through this regex
SSLSNIFF_CONTAINER_EXCLUDE_FILTER	-	Exclude any container from monitoring by this regex
DOCKER_API_VERSION	-	Compatibility environment variable (if required). This should be the max supported version by the docker daemon on ECS. This can be gotten from the output of, `docker version | grep API`
LOGGING_LEVEL	info	sniffer logging level. Set among trace|debug|info|error
CUSTOM_LIBSSL_1_PATH **CUSTOM_LIBSSL_3_PATH**	-	Define non-default openssl ver 1/3 lib paths, reference is debian paths by default. Example for RHEL, CUSTOM_LIBSSL_1_PATH=/usr/lib64/libssl.so.1.1.1k

Procedure

1. Create a task definition for sslsniffer
    a) Populate the required environmental variables as above
    b) Provide the right mounts,
        i. Docker daemon interface via mount of /var/run/docker.sock
        ii. Capabilities of “SYS_ADMIN” and “SYS_PTRACE”
        iii. Process space of host via pidMode=host
        iv. Privilege mode of execution via privileged=true needed for host equivalent access
2. Create another service of type DAEMON under the cluster to be monitored

Reference task definition:
https://sample-config.appsentinels.ai/appsentinels-deployment/aws-ecs/sslsniffer-daemon-ecs-task-def.json

Note

- **Please note that older Linux kernels (<5.10) do not have complete support for eBPF tracing by default. It is recommended to use 5.10+**

- **eBPF based sniffers cannot be supported on FARGATE due to runtime restrictions.**

Sample task definition

```
{
    "containerDefinitions": [
        {
            "name": "sniffer",
            "image": "appsentinels/sslsniffer:latest",
            "essential": true,
            "environment": [
                {
                    "name": "REMOTE_CONTROLLER_SERVER_ENDPOINT",
                    "value": "https://edge-controller:9004/mergedlog"
                },
                {
                    "name": "DOCKER_API_VERSION",
                    "value": "1.44"
                },
                {
                    "name": "INSTANCE_NAME",
                    "value": "http-app"
                }
            ],
            "mountPoints": [
                {
                    "sourceVolume": "docker-socket",
                    "containerPath": "/var/run/docker.sock",
                    "readOnly": false
                }
            ],
            "linuxParameters": {
                "capabilities": {
                    "add": [
                        "SYS_ADMIN",
                        "SYS_PTRACE"
                    ],
                    "drop": []
                }
            },
            "privileged": true,
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "/ecs/sslsniffer-daemon",
                    "mode": "non-blocking",
                    "awslogs-create-group": "true",
                    "max-buffer-size": "25m",
                }
            },
            "systemControls": []
        }
    ],
    "volumes": [
        {
            "name": "docker-socket",
            "host": {
                "sourcePath": "/var/run/docker.sock"
            }
        }
    ],
    "pidMode": "host",
}
```


Kubernetes Daemon Set Sniffer

Similar to EC2 daemonset, the SSL sniffer can deployed as a daemon set onto any kubernetes cluster.

Configurations

Environment	Default	Description
REMOTE_CONTROLLER_SERVER_ENDPOINT	-	Edge controller endpoint with scheme, host and path. Eg: https://edge-controller:9004/mergedlog
INSTANCE_NAME	Unknown	Instance name for visibility. Useful to identify sensors on the dashboard, mention something meaningful here
SSLSNIFF_CONTAINER_INCLUDE_FILTER	-	By default all the containers are monitored, but specify only particular container through this regex
SSLSNIFF_CONTAINER_EXCLUDE_FILTER	-	Exclude any container from monitoring by this regex
DOCKER_API_VERSION	-	Compatibility environment variable (if required). This should be the max supported version by the docker daemon on ECS. This can be gotten from the output of, `docker version | grep API`
LOGGING_LEVEL	info	sniffer logging level. Set among trace|debug|info|error
CUSTOM_LIBSSL_1_PATH **CUSTOM_LIBSSL_3_PATH**	-	Define non-default openssl ver 1/3 lib paths, reference is debian paths by default. Example for RHEL, CUSTOM_LIBSSL_1_PATH=/usr/lib64/libssl.so.1.1.1k

Reference Spec:
https://sample-config.appsentinels.ai/appsentinels-deployment/sslsniffer/k8/sniffer-daemonset.yaml

RBAC (Role-Based Access Control)

For sniffer to scan the properties of pods, it requires the correct accesses.

1. ServiceAccount (sslsniffer-sa)
   SSL sniffer will use this service account to interact with the Kubernetes API securely. It's needed if your app wants to list pods, etc.

2. ClusterRole (sslsniffer-role)

   This will defines **what actions** the service account can perform. It gives **read-only access** (get/list/watch) to:

   - **pods**, **pods/status**, **pods/log** – to inspect and monitor pods
   - **nodes** – possibly for identifying host-level details
   - **services** – to correlate endpoints
   - **events** – to react to what's happening in the cluster
   - **apps resources** – like daemonsets, deployments, replicasets, statefulsets (to watch application deployments)

3. RoleBinding (sslsniffer-role-binding)
   This connects the service account to the cluster role for the ssl sniffer.

All the above properties can be referenced in the below link:
https://sample-config.appsentinels.ai/appsentinels-deployment/sslsniffer/k8/role.yaml

Procedure

1. Create service account and configure the RBAC
2. Deploy the daemonset using the above spec
3. If required configure the filters for specific pod sniffing

Ingress controller Sniffer

Based on deployment, it is efficient to sniff on the ingress controller instead of all the application PODs. This provides for a single point of contact. The below configuration and procedure ensure that the sniffer daemon set is deployed only on nodes that ingress controller POD is scheduled.

Configurations

Same as earlier, but with the below exception of setting this env filter. This will ensure only ingress POD will be searched for.

```
env:

- name: SSLSNIFF_CONTAINER_INCLUDE_FILTER
  value: "nginx-ingress"
```

Procedure

Reference Specification: https://sample-config.appsentinels.ai/appsentinels-deployment/sslsniffer/k8/sniffer-daemonset-ingress.yaml

1. Create the service account and provide permissions as earlier
2. Please note down the label used for deploying ingress controller. This can be figured from the below command, or equivalent,

   kubectl get pods -n ingress -l app.kubernetes.io/name=ingress-nginx --show-labels

3. Update the ingress label under **affinity.podAffinity.requiredDuringSchedulingIgnoredDuringExecution.labelSelector** of the reference specification
4. Update the namespace selector for the affinity configuration to the namespace ingress is running on. This can be updated at, **Affinity.podAffinity.requiredDuringSchedulingIgnoredDuringExecution.namespaceSelector**
5. Apply the daemonset specification to deploy

Host based sniffer

SSL sniffer can be deployed directly on host (compared to a container) to sniff traffic of host processes.

Configurations

Environment	Default	Description
REMOTE_CONTROLLER_SERVER_ENDPOINT	-	Edge controller endpoint with scheme, host and path. Eg: https://edge-controller:9004/mergedlog
INSTANCE_NAME	Unknown	Instance name for visibility. Useful to identify sensors on the dashboard, mention something meaningful here
SSLSNIFF_INCLUDE_PROC_FILTER	-	By default all the host processing are attempted to be monitored, but specify fine tune this selection based on this regex
SSLSNIFF_EXCLUDE_PROC_FILTER	-	Exclude any process from monitoring by this regex
SNIFF_LOCAL_ONLY	-	Set this to “true” to enable host sniffing
LOGGING_LEVEL	info	sniffer logging level. Set among trace|debug|info|error
CUSTOM_LIBSSL_1_PATH **CUSTOM_LIBSSL_3_PATH**	-	Define non-default openssl ver 1/3 lib paths, reference is debian paths by default. Example for RHEL, CUSTOM_LIBSSL_1_PATH=/usr/lib64/libssl.so.1.1.1k

Procedure

For RHEL, rpm is provided here,
https://sample-config.appsentinels.ai/appsentinels-deployment/sslsniffer/redhat/appsentinels-sslsniffer-1.0.1-1.x86_64.rpm

1. Install the rpm
2. The service ‘appsentinels-sslsniffer’ will be started by default
3. Perform fine tuning via environmentals by updating
   /etc/sysconfig/appsentinels-sslsniffer
4. Restart the service post updating
   systemctl restart appsentinels-sslsniffer.service

Verify Deployment

1. Please verify if the daemon container is running correctly
2. If traffic is being sniffed correctly, the sensor should be visible on the dashboard under sensors

