Arquitectura de SO

# Arquitectura de Security Onion

Security Onion se trata de una distribución **GNU/Linux** para constituir un **SOC** *(Security Operations Center)* perfectamente capacitado para detectar, analizar y responder ante amenazas de seguridad.

Para ello implementa un **SIEM** *(Security Information and Event Management)* para recoger y correlacionar los registros de logs y eventos de seguridad capturados para detectar amenazas. Para este fin se usa el **Stack de Elastic** como **SIEM** de **Security Onion**.

:::tip[Aplicaciones de la pila de Elastic] - **ElasticSearch**: *motor de búsqueda y análisis en tiempo real* - **Logstash**: *ingesta y transformación de datos* - **Kibana**: *interfaz gráfica de visualización de datos almacenados en* ***ElasticSearch***. - **ElasticAgent**:*recolección de logs y monitoreo en tiempo real*. :::

Además como **IDS/IPS** implementa la funcionalidad de detección y prevención de intrusiones con **Suricata** y **Zeek**.

* **Suricata**, *analiza el tráfico de red y genera avisos sobre la actividad que detecta como sospechosa*.
* **Zeek**, *captura y analiza el tráfico permitiendo generar registros sobre las incidencias*.

Además permite la gestión de **endpoints** con **Fleet**, el cual utiliza agentes de \*\*elastciAgent\*, permitiendo análisis y búsquedas centralizadas.

## Implementación de la arquitectura de SO

Security Onion ejecuta como base un sistema GNU/Linux basado en **Oracle Linux (RHEL)** que a su vez utiliza contenedores **Docker** para poder empaquetar y gestionar de la mejor manera las aplicaciones que implementa el **SOC**, aportando flexibilidad y escalabilidad, además de establecer un sistema aislado de gestión donde cada servicio se administra de forma independiente evitando interferencias entre servicios y problemas futuros derivados de las actualizaciones de cada uno. Además, para la orquestación de este entorno multicontenedor se utiliza \*\*docker compose\*.

### Docker

Se trata de un sistema de contenedores que permite construir elementos estancos e independientes para ejecutar servicios evitando conflictos entre los diferentes servicios.

#### Docker compose

Los contenedores de **Docker** pueden requerir de configuraciones personalizadas diferentes a las imágenes disponibles en el **docker hub container** *(como imágenes propias realizadas a través de dockerfiles)*, y junto a las del docker hub, se pueden implementar de una forma sencilla a través de **docker-compose** y el uso de ficheros \***Yaml**, de esta forma se puede trabajar en un entorno multi-contenedor de una forma sencilla.

### Principales scripts de Docker para SO

**Security Onion** dispone de unos scripts básicos para interaccionar con los contenedores.

Listado de scripts para realizar funciones con los contenedores de Security Onion

* so-setup: el script inicial para desplegar Security Onion, actua como guía del proceso de configuración e instalación.
* so-docker: script para la gestión de los contenedores docker que implementan los servicios en Security Onion.

sudo so-docker ps  
sudo so-docker start / stop / restart

* so-config: para modificar la configuración de herramientas del sistema como suricata o elastic stack desde la terminal.
* so-elastic: controlar y gestionar ElasticSearch
* so-roles: gestión de las políticas de Security Onion, como los permisos de los usuarios.
* so-prepare: para la instalación de de componentes adicionales de Security Onion
* so-status: muestra el estado de todos los contenedores en ejecución en Security Onion
* so-update: realiza la actualización de todos los componentes a la última versión. IMPORTANTE
* so-logging: información y recapitulación de los logs del OS.

#### Comandos adicionales para usar con docker y los contenedores

**Detener todos los contenedores**

sudo docker stop $(sudo docker ps -q)

**Inicializar un contenedor concreto**

sudo docker container "nameContainer/id"

**Consultar los logs del sistema**

sudo docker logs "containerName/id"

### Salt

La infraestructura de **SaltStack** se basa en una arquitectura cliente/servidor para la **administración remota** que actua como el backbone de **Security Onion** para la interacción entre los diferentes **nodos** de una **arquitectura distribuida**, de esta forma, mediante **Salt** se pueden utilizar diferentes nodos sin necesitar de intervención manual en cada máquina.

#### Estructura de la arquitectura

Como se ha comentado, basa su funcionamiento en una arquitectura Cliente / Servidor con una nomenclarura propia.

* **Salt Master**: es el nodo maestro que administra la red distribuida y controla los nodos.
* **Nodos o Salt Minions**: ejecutan los comandos que reciben desde el maestro.

El **master** es el **nodo principal**, que puede estar en el modo **Standalone** y **Evaluate**, en *resto de nodos* actuan los **minions**.

Además del master y de los minions hay que tener en cuenta los siguientes elementos:

* **Grains**: es la información detallada de cada minion (CPU, RAM…)
* **Pillars**: datos importantes y configuraciones personalizadas para los minions.
* **States**: aplican las configuraciones y estados automatizados para los minions.

#### Funcionamiento de Salt

Listado de las principales funciones realizadas por Salt

* **Automatización de la configuración de los minions** a través del master.
* **Sincronización de las reglas** de detección de intrusiones y de las **configuraciones de seguridad** implementadas.
* **Distribución** de las **actualizaciones y parches** de seguridad a toda la estructura distribuida.
* **Orquestación** para la gestión de **Docker** y de los **contenedores implementados** para los diferentes servicios.
* Realiza la **monitorización** del **sistema** acorde a las políticas.

#### Ficheros importantes de Salt

Los ficheros de configuración están escritos en formato **YAML**, destacan los siguientes: - **/etc/salt/master**: configuración del master - **/etc/salt/minion**: configuración de los minions - **/opt/so/saltstack/local**: estado y configuración - **/opt/so/saltstack/local/pillar**: datos sensibles de pillar - **/etc/salt/grains**: información de los grains

#### Ejemplo de uso de comandos de Salt

**Listar los minions**

sudo salt-key -A

**Enviar un comando a todos los minions**

sudo salt '\*' test.ping

**Consultar los Grains de un minion**

sudo salt 'identificador-minion' grains.items

**Aplicar un estado a un minion**

sudo salt 'indentificador-minion' state.apply 'stateName'

**Reiniciar todos los minons**

sudo salt '\*' system.reboot

**Verificar permisos configurados en el master**

sudo salt-run pillar.items

**Editar reglas servicio, por ejemplo Suricata**

sudo nano /opt/so/saltstack/local/salt/suricata/rules/custom.rules

**Aplicar las reglas editadas de Suricata a todos los minions**

sudo salta '\*?' state.apply suricata

**Reinicar Suricata en los minions para aplicar las reglas**

sudo salt '\*' cmd.run 'systemctl restart suricata'