**Suricata**

|  |  |  |  |
| --- | --- | --- | --- |
| **DESC** | | | |
| IDS/IPS | | | |
| **FILES** | | | |
| /etc/suricata/ | suricata.yaml | Configuration | |
|  | rules/ | Règles | |
|  | classification.config | Classement des niveaux d’alertes en fonction du classtype de la règle  Se retrouve dans alert\_category au niveau de l’alerte | |
|  | reference.config | Permet de définir des alias à utiliser avec l’option référence de la règle | |
|  | threshold.config | Valeur globale de seuil ou de limite de génération d’alerte  + les règles supress | |
| /var/ | Log/suricata/ | Suricata.log | Logs internes à Suricata |
|  |  | Stats.log | Statistiques de performance Suricata |
|  |  | Fast.log | Logs des alertes rapides |
|  |  | Eve.json | Logs des alertes et évènements en JSON |
|  |  | http.log | Logs des headers HTTP |
|  |  | X.log | Si défini dans la configuration |
|  | /run/suricata.pid | PID du démon si renseigné dans la configuration | |
| **QUICK START** | | | |
| * Suricata -i <interface> * Afficher header http dans l’alerte * Afficher les netflows * Investiguer les Faux positifs | | | |

**Généralités :**

* **Thread modules :**
  + Packet acquisition : lis les paquets entrants du réseau
  + Decode & stream application layer
    - Track les flows
    - Reconstruction du flux fragmenté
    - Inspection de la couche applicative
  + Détection
  + Outputs
* **Acquisition :** 
  + RSS du NIC
  + Acquisition par le noyau (PF\_RING, PCAP…)
  + Optimisation dans Suricata (cpu\_afinity…)
* **Decode & stream app layer :** 
  + Suricata réassemble les fragments pour obtenir un paquet entier
  + Stream : TCP connexions faisant parti d’un flow
  + Flow : tout paquet possédant le même Tuple (protocole, @src, @dest, src\_port, dest\_port)

C’est Suricata qui réassemble les flux (et les fragments aussi)

Via les paramètres de flow dans les règles, on peut établir des corrélations : Nombre d’occurrences, Valeur seuils avant déclenchement d’une alerte…d’un même flux.

Pour un flow non répertorié, Suricata crée un flow.

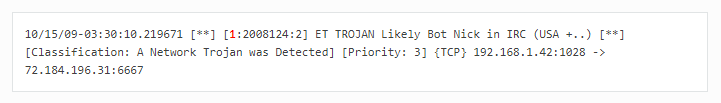
Risque de DOS Suricata : la création d’un flow étant très couteuse, on inonde l’IDS de plein de flow nouveau et on envoie une attaque. On peut donc préalablement réserver de la mémoire pour des flows pour contrer ça.

* **Détection :**
  + MPM : multi pattern matcher. Dans une signature, il peut y avoir plusieurs "content :…", soit plusieurs pattern. Suricata en utilise un comme étant le fast\_pattern, il sera utilisé en premier pour établir un match.
  + Possibilité d’utiliser Hyperscan : Librairie haute performance d’analyse REGEX
  + Possibilité d’utiliser des groupes de signatures afin d’optimiser le traitement des règles : Plus y a de groupes, plus c’est performant mais plus on utilise de RAM
* Différence entre content modifier et sticky buffer
* Alerte si protocoles non communs etc…, blacklist SSH etc… voir règles
* Possibilité de mettre des valeurs de seuils (Une alerte est générée après X fois ou bien on définit une limite afin d’éviter le flood d’alertes..), dans la configuration de manière générale ou bien via certains mots-clefs d’une règle en particulier.
* Suricata-update (>= 4.1) ou Oinkmaster pour la MAJ de règles et un Cron pour automatiser.
* Socket UNIX permet de dialoguer avec Suricata en live (ex on peut reloader les règles via la socket UNIX au lieu d’envoyer un signal USR2)

Cet échange se fait en JSON

* Possibilité de dropper des paquets après 10 connections à la minute via rate\_filter dans global\_threshold
* Possibilité de mettre à jour une liste d’IPs trusted ou avec mauvaise réputation : les IPs reputation.
  + Un fichier catégorie
  + Un fichier pour dire les IPs trusted ou compromises
* Possibilité de mettre deux fois eve.log dans output de la configuration pour stocker en JSON deux types d’évènements différents
* Scripts LUA peuvent être utilisés dans :
  + Output pour écrire un output personnalisé
  + Ecriture de règles
* Possibilité de stocker les fichiers contenus dans le corps d’un paquet via un filestore ;

**Format d’une alerte :**



[**GID** :SID :REV]

**Signaux :**

|  |  |
| --- | --- |
| **Signal** | **Comments** |
| SIGUSR2 | Live rule reload |
| SIGHUP | Close and reopen log file |

**Fichier de configuration :**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Directive** | **Comments** | | | | | | | |
| **GENERALE** | | | | | | | | |
| **Pid-file : <path>** |  | | | | | | | |
| **action-order :**  **-pass**  **-drop**  **-reject**  **-alert** | Ici, les règles pass ne soulève pas d’alertes donc ce sont des règles qui peuvent annuler d’autres règles  On peut faire un fichier d’exclusion basé sur des pass | | | | | | | |
| **Include : <path.yaml>** | Inclusion de fichiers de configuration dans la configuration générale | | | | | | | |
| **default-log-dir : <path>** | Output de Suricata | | | | | | | |
| **outputs :** | Activer uniquement les outputs nécessaires | | | | | | | |
|  | **Fast.log** | Alertes au format syslog | | | | | | |
|  | **Eve-log** | Alertes + events au format JSON | | | | | | |
|  |  | **Append** | | | | | | Quand restart de Suricata, on overwrite pas le fichier |
|  |  | **Alert.payload** | | | | | | Nouveau champs dans JSON : payload en Base64 |
|  |  | **Alert.payload-printable** | | | | | | Idem mais payload en format affichable |
|  |  | **flow** | | | | | | Enregistre le flow dans sa globalité |
|  |  | **netflow** | | | | | | Plus intéressant, enregistre toutes les communications d’un flow |
|  | **Unified2** | si utilisation de Barnyard2 | | | | | | |
|  | **http-log** | log des headers http | | | | | | |
|  | **dns-log** | log des requêtes DNS | | | | | | |
|  | **pcap-log** | Log de tous les paquets inscrits dans Suricata | | | | | | |
|  | **alert-debug** | Log alerte très verbeux : permet d’éliminer les Faux Positifs. Métadonnées + alerte+ payload raw dans alert-debug.log | | | | | | |
|  | **alert-prelude** | Si utilisation de Prelude | | | | | | |
|  | **stats** | Statistiques de performances Suricata | | | | | | |
|  | **syslog** | Envoie de toutes les alertes et événements à Syslog | | | | | | |
|  | **drop.log** | Log des paquets droppés par Suricata | | | | | | |
|  | **file-store** | Stockage des fichiers extraits par Suricata | | | | | | |
| **Logging :** | Log du moteur Suricata  Se retrouve dans suricata.log | | | | | | | |
| **FONCTIONNEMENT** | | | | | | | | |
| **Libhtp :** | Body-limit | | | | Taille d’inspection maxi du body http à inspecter  0 = aucune limite  Mettre à 0 si on stocke les fichiers | | | |
| **PERFORMANCE** | | | | | | | | |
| **runmode** | Single|auto|autofp|workers  Workers pour de meilleures performances : un seul thread s’occupe de tout | | | | | | | |
| **max-pending-packets** | Nombre de paquets simultanés que l’engine peut traiter.  Mettre à 1024 | | | | | | | |
| **Default\_packet\_size** | Taille des paquets pour une optimisation maximum  Mettre à 1514 | | | | | | | |
| **mpm-algo** | ac|hs|ac-bs|ac-ks|…  Hyperscan le meilleur choix | | | | | | | |
| **detect** | **profile** | | | Low|medium|high|custom  Les signatures sont splittés en groupes  Higher : meilleur performances mais plus de RAM | | | | |
|  | **sgh-mpm-context** | | | Auto|single|full   * Full : Chaque groupes possède son propre contexte MPM * Single : un seul MPM pour tous les groupes | | | | |
|  | **Inspection-recursion-limit** | | | | | | Permet d’éviter les boucles infinies (si bug de Suricata) | |
| **set-cpu-affinity** | Permet d’affecter des thread modules à des CPU particuliers cf. plus bas | | | | | | | |
| **detect-thread-ratio : 1.5** | Nombre de threads detect : 1.5 x #CPU/Core | | | | | | | |
| **defrag :** | Pour le réassemblage des paquets fragmentés avant analyse | | | | | | | |
|  | Max-frags : | |  | | | | | |
|  | Prealloc : | |  | | | | | |
|  | Timeout : | |  | | | | | |
| **Pfring :** | Cluster-id : | | Doit être unique pour une instance de Suricata | | | | | |
|  | Cluster-type : | | Cluster\_round\_robin|cluster\_flow | | | | | |
| **SECURITE** | | | | | | | | |
| **Run-as :**  **user : suri**  **group : suri** |  | | | | | | | |
| **Flow :** | Flow = flux possédant le même tuple  Permet d’éviter un flood de nouveaux flow pouvant bloquer l’IDS, la création d’un flow étant très couteux.  Si on atteint les limites, les flows déjà présents auront un timeout très agressif. | | | | | | | |
| **Stream :** | Stream = connexion TCP | | | | | | | |
|  | midstream | | | | | Lorsqu’on lance Suricata, savoir si on inspecte quand même les connexions pris à la volée : on n’a pas assisté au 3WHS | | |
|  | Async\_oneside | | | | | Si les 3WHS arrivent dans le désordre | | |

**Runmodes de Suricata :**

Plusieurs blocs :

* Threads (de manière générale)
* Module-threads (une activité spécifique ex : decode, detect, output…)
* Les queues (intermédiaire entre des threads)

Un runmode est une manière d’arranger tout ça.

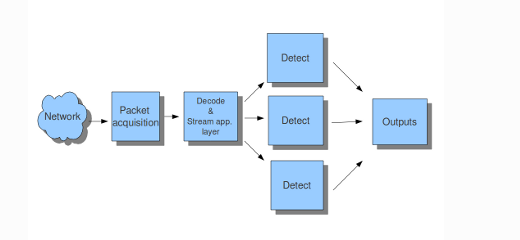
Runmodes :

* Single : 1 seul thread
* Auto : multi-threaded : chaque **paquet** possède son propre thread detect.
* Autofp : Idem sauf que chaque **flow** possède son propre thread detect.
* Workers : meilleurs performances : chaque thread font tout de l’acquisition à l’output, ce mode dépend donc de la méthode de capture des paquets, puisque un flow doit être assigné à un unique thread : on doit donc utiliser une méthode qui privilégie un load balancing des paquets suivant un hash des 5-tuples (@dest, @src, ports…) soit AF\_PACKETS ou PF\_RING

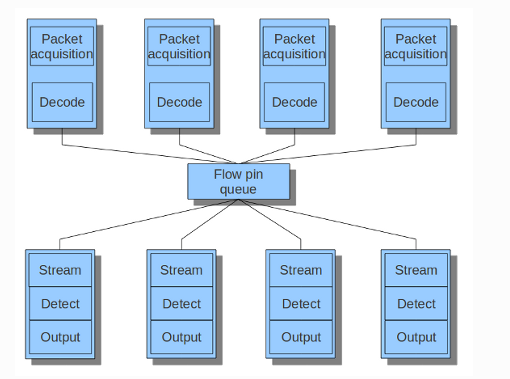
Ce hash doit de plus être symétrique : (@src, @dest…) doit envoyer dans la même file que (@dest, @src…) pour envoyer le flow au même thread

|  |  |  |
| --- | --- | --- |
| **NIC 🡪** | **Méthode de capture du noyau** | **🡪 Mode Suricata** |
| RSS | PCAP | Single |
| … | PFRING | auto |
|  | AF\_PACKET | autofp |
|  | NETMAP | workers |

Mode par défaut :



Mode autofp :



**Cpu-affinity :**

