**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 | |
| /var/ | Log/suricata/ | Suricata.log | Logs internes à Suricata |
|  |  | Stats.log | Statistiques post Suricata |
|  |  | Fast.log | Logs des alertes rapides |
|  |  | Eve.json | Logs des alertes en JSON |
|  |  | http.log |  |
|  | /run/suricata.pid | PID du démon si renseigné dans la configuration | |
| **QUICK START** | | | |
| * Suricata -i <interface> | | | |

**Généralités :**

* 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.
* Flow : tout paquet possédant le même Tuple (protocole, @src, @dest, src\_port, dest\_port)

C’est Suricata qui réassemble les flux.

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.

* 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.
* Signaux :
  + USR2 pour reloader les règles
  + SIGHUP pour la rotation de logs.
* 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)

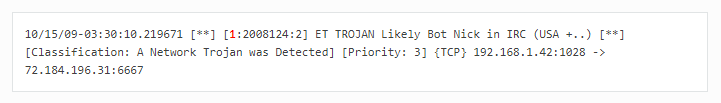
**Signaux :**

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

**Fichier de configuration :**

|  |  |
| --- | --- |
| **Directive** | **Comments** |
| **GENERALE** | |
|  |  |
| **OBLIGATOIRE** | |
|  |  |
| **OPTIMISATION** | |
|  |  |
| **SECURITE** | |
|  |  |

**Format d’une alerte :**



[**GID** :SID :REV]

**Runmodes de Suricata :**

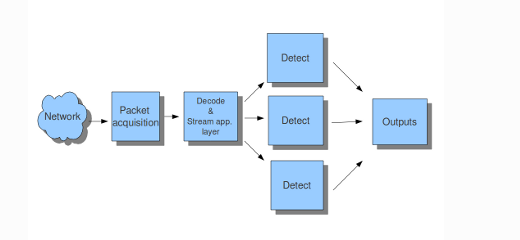
Runmodes : plusieurs blocs : threads (de manière générale), module-threads (une activité spécifique ex : decode, detect, output…) et les queues (intermédiaire entre des threads).un runmode est une manière d’arranger tout ça.

* 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 :

