# Sintaxis de reglas de Falco y reglas para kernel

# Ficheros de reglas y carpetas predefinidas

# Ficheros y carpetas predefinidas

**/etc/falco/**

**falco_rules.yaml**: default rules, overwritten on Falco upgrade

**falco_rules.local.yaml**: never overwritten, put your rules here

**k8s_audit_rules.yaml**: default Kubernetes related rules

**falco.yaml**: General Falco configuration

**rules.available/application_rules.yaml**: Move to rules.d to activate

**rules.d/**: Files in this dir will be processed as rules in alphabetical order

# Sintaxis de reglas de Falco

https://falco.org/docs/rules/

# Ficheros de reglas Falco

Un fichero de reglas de Falco usa formato *YAML* y contiene tres tipos de elementos:

- **rule** : regla, ondición bajo la cual se dispara una alerta con una salida informative a enviar a la salida.
- **macro** : condición booleana a evaluar para combinar en otras macros y reglas, de forma que se puedan reutilizar.
- **list** : lista de elementos para utilizar en reglas, macros u otras listas.

# YAML

https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html

Acostúmbrate a la sintaxis de YAML:

- Debes usar doble espacio para identar (los editors de código convertirán tabs correctamente por ti).
- Un error de identación o espacios de más provocarán error.
- Los nuevos elementos comienzan con - (guión medio).
- Usar \ al final de una linea para continuar en la siguiente.
- Usar > al principio de un valor para escribir en las siguientes lineas sin tener que añadir \ al final.

# Lists

```
- list: sensitive_file_names
  items: [/etc/shadow, /etc/sudoers, /etc/pam.conf,
/etc/security/pwquality.conf]
```

Especificando `append:true` añadimos elementos a una lista anterior.

```
- list: sensitive_file_names
  append: true
  items: [/etc/ssh/ssh_host_ecdsa_key, /etc/ssh/ssh_host_rsa_key]
```

# Macros

```
- macro: sensitive_files
  condition: >
    fd.name startswith /etc and
    (fd.name in (sensitive_file_names)
     or fd.directory in (/etc/sudoers.d, /etc/pam.d))

- macro: sensitive_files
  append: true
  condition: or fd.name startswith /root
```

https://falco.org/docs/rules/

# Rules

Una regla es un nodo en el fichero YAML que contiene los siguientes atributos:

- **`rule`**: nombre único de la regla
- **`condition`**: expression booleana que filtra los eventos entrantes.
- **`desc`**: descripción larga de qué detecta la regla
- **`output`**: mensaje a enviar por el canal de notificación cuando ocurre una detección del evento.
- **`priority`**: representa el nivel de importancia del evento detectado: `"emergency"`, `"alert"`, `"critical"`, `"error"`, `"warning"`, `"notice"`, `"informational"`, o `"debug"`.

Atributos opcionales: `enabled`, `tags`, `warn_evttypes`, `skip-if-unknown-filter`, y `required_engine_version`.

# Tipo de evento y filtros conocidos

- Para optimizer el rendimiento, Falco intenta agrupar las reglas por tipo de evento, para ello la condición debe tener **al principio** un operador **`evt.type=`** antes que ningún operador de negación (por ejemplo `not` o `!=`).
- De lo contrario se mostrará por la salida un mensaje de aviso, a menos que la regla especifique:
  `warn_evttypes: true`
- Si en la salida se especifica un atributo desconocido, se muestra un mensaje de aviso a menos que se establezca
  `skip-if-unknown-filter: true`

# Comillas dobles y simples

Las reglas puede contener carácteres especiales como paréntesis, espacios, etc. Por ejemplo, el nombre de un proceso puede ser (systemd) incluidos los paréntesis. Para especificar estos casos usaremos comillas dobles

```
proc.name="(systemd)"
```

Cuando se incluyan elementos en una lista, debemos asegurarnos que las comillas dobles no son interpretadas directamente como parte del YAML rodeándolas de comillas simples:

```
- list: systemd_procs
  items: [systemd, '"(systemd)"']
```

# Rule: MySQL genera una shell

```
- rule: Database spawns a shell
  condition: >
    proc.pname in (db_server_binaries) and spawned_process
    and not proc.name in (db_server_binaries)
    and not postgres_running_wal_e
  output: >
    Database-related program spawned process other than itself
    (user=%user.name program=%proc.cmdline parent=%proc.pname)
  source: syscall
  desc: >
    Database-server program spawned a new process other than itself.
    This shouldn't occur and is a follow on from some SQL injection attacks.
  priority: WARNING
  tags: [process, database]
```

# Campos soportados

System Calls: https://falco.org/docs/rules/supported-fields/

- **fd**
- **proc** & **thread**
- **evt**
- **user** & **group**
- **syslog**

Containers, Kubernetes, Mesos, JSON

- **container**
- **k8s**
- **mesos**
- **jevt**

Kubernetes audit: https://falco.org/docs/event-sources/kubernetes-audit/

- **ka**

Grammar reference in code:
https://github.com/falcosecurity/falco/blob/master/userspace/engine/lua/parser.lua

# Operadores

A **pmatch** B : *A partial match any element in list B, returns bool*

A **intersects** B : elements of A that also are in B, returns list

A **in** B : is element A in list B? returns a bool

A **endswith** B : string A ends with string B, returns bool

A **startswith** B : string A starts with string B, returns bool

A **glob** B : string A checks with expandable pathname B, return bool

A **icontains** B : string A contains string B, case insensitive, returns bool

A **contains** B : string A contains string B, case matter, returns bool

A **exists** : string A is defined and not empty, returns bool

**not** A : negation of bool value A

**> < >= <= != =** : number comparison

# Validar reglas

Probar reglas en directorio por defecto:

```
$ falco -L
```

Probar reglas en directorio por defecto(no mostrar lista complete después):

```
$ falco -L >/dev/null
```

Docker / Kubernetes probar reglas por defecto:

```
$ docker exec falco sh -c "falco -L >/dev/null"
$ kubectl exec -it $falco_pod -n falco -- sh -c "falco -L
>/dev/null"
```

Probar fichero individual de reglas (debe incluir todas sus listas y macros)

```
$ falco -V filename1.yaml -V filename2.yaml
```

Probar todos los ficheros *.yaml usando imagen de Falco sin instalar en kernel

```
$ docker run -v `pwd`/rules:/rules --rm -i -t \
  falcosecurity/falco-no-driver:latest falco \
  `ls rules/*.yaml | xargs -I {} echo -V {} | xargs`
```

# Recargar reglas

Reiniciar servicio con Falco instalado en host

```
$ sudo /etc/init.d/falco restart
```

Recargar reglas con Falco ejecutado por Docker

```
$ docker restart falco
```

Recargar reglas en Kubernetes:

```
$ ./rules2helm.sh falco_rules.local.yaml
>rule_update_falco.yaml
$ helm upgrade falco stable/falco \
    -f rule_update_falco.yaml \
    --namespace falco \
    --wait --timeout 40s
```