HIDS Falco

Falco (https://falco.org) és un programa que monitoritza l'activitat d'un sistema (o més concretament, que detecta totes les crides al sistema Linux efectuades per tots els processos executats en cada instant).

Una crida al sistema és una funció proporcionada pel kernel que pot ser invocada per qualsevol procés que en vulgui fer ús. L'accés a determinats recursos del sistema requereix l'execució de codi en mode privilegiat; és per això que el kernel ofereix un conjunt de crides al sistema (a mode d'"API") que qualsevol programa pot utilitzar per accedir a aquests recursos (memòria, CPU, disc, etc) d'una forma controlada. Per tant, podem dir que el kernel actua d'intermediari entre els programes i el hardware, oferint un punt centralitzat on ambdós extrems es poden comunicar d'una forma coneguda i estructurada fent ús del conjunt establert de "crides al sistema" disponibles.

Alguns exemples de "crides al sistema" són *write()*, utilitzada per escriure una dada en un dispositiu de sortida (com pot ser una pantalla o un disc), *read()*, utilitzada per llegir una dada en un dispositiu d'entrada (com un teclat o un disc), *connect()*, utilitzada per obrir un port per connectar-se en mode client a un altre extrem de la xarxa, etc. Algunes crides comunes són:

open()/openat() : Obre un fitxer. Retorna un id numèric únic pel fitxer anomenat "file descriptor" (fd)
NOTA: Els fd nº 0, nº 1 i nº2 són especials: el primer representa stdin (el teclat), el segon stdout i el tercer stderr
access() : Comprova els permisos d'un fitxer (indicat pel seu fd com a primer paràmetre)
NOTA: Una crida més general seria ***fstat()*** o també ***statfs()***
read() : Llegeix un fitxer (indicat pel seu fd com a primer paràmetre)
write() : Escriu en un fitxer (indicat pel seu fd com a primer paràmetre)
lseek() : Mou el cursor de memòria al llarg del fitxer (per poder llegir-hi o escriure-hi a la nova posició)
fchmod() : Canvia els permisos d'un fitxer (indicat pel seu fd com a primer paràmetre)
NOTA: També està ***fchown()***, que canvia el propietari
close() : Tanca un fitxer (indicat pel seu fd com a únic paràmetre)
unlink() : Esborra un fitxer (indicat pel seu fd com a primer paràmetre)
chdir() : Estableix el directori de treball del procés en qüestió
socket() : Obre un socket local. Retorna també un "file descriptor" (fd) identificador del socket
connect() : Obre una conexió des d'un socket local amb un socket remot (normalment per fer de client)
bind() : Activa un socket local (pas previ imprescindible per posar-lo a l'escolta tot seguit amb ***listen()***)
setsockopt() : Defineix diferents opcions per un determinat socket. També està ***accept()***
sendto() : Envia dades a un socket remot. També es pot usar "*write()*"
recvfrom() : Rep dades en un socket local. També es pot usar "*read()*"
mmap()/munmap() : Carrega/descarrega dades a/de la memòria catxé de la RAM. També està ***brk()***
execve() : Sol ser la primera crida que apareix en un programa. Serveix per "autoexecutar-se"
system() : Executa un shell script.
fork()/clone() : El programa es clona a sí mateix (normalment això es fa per convertir-se en dimoni)
kill() : Envia una senyal a algun altre procés

En tot cas, una llista de les crides al sistema més usades (amb una breu explicació de què fan i com funcionen) es troba a https://sysdig.com/blog/fascinating-world-linux-system-calls, tot i que una llista molt més completa es troba a https://github.com/gregose/syscall-table (o consultant *man syscalls*). No obstant, si es vol accedir a la documentació completa per saber què fa, quins paràmetres té, quins valors de retorn té cadascuna de les crides al sistema existents, el millor és consultar les pàgines del manual (concretament la secció 2, així: *man 2 read*, per exemple).

Cal tenir en compte, d'altra banda, que abans que una crida al sistema demanada per un programa comenci a processar-se, el kernel interromp l'execució d'aquest programa en qüestió en aquell punt durant el període en què s'estigui realitzant la crida; un cop finalitzada l'acció implementada per aquesta crida, és llavors quan el kernel "allibera" l'execució del procés i permet que aquesta continui des del punt del seu codi en què es va fer la crida.

No obstant, Falco no només monitoritza, sinó que, a més, a partir d'aquesta activitat observada és capaç de detectar comportaments anòmals gràcies a determinades regles preestablertes que identifiquen els events/valors fora de lloc. Per tant, Falco pot servir per controlar contínuament l'activitat d'aplicacions, contenidors, sistemes amfitrions, xarxes... i generar-ne alertes si aquesta activitat coincideix en un moment donat amb alguna de les regles definides. Per exemple, Falco pot detectar fàcilment coses "sospitoses" com, entre d'altres:

*L'execució d'un shell dins d'un contenidor
*La generació d'un procés fill d'un tipus inesperat per part d'un procés servidor
*La lectura inesperada d'un fitxer "sensible" (com ara "/etc/shadow")
*L'escriptura dins de "/dev" d'un fitxer de tipus "non-device"
*La realitzatció d'una connexió de xarxa per part d'un binari estàndard del sistema (com ara *ls*)

Per instal.lar Falco, només cal seguir les instruccions indicades a https://falco.org/docs/getting-started/installation, les quals consisteixen bàsicament en afegir un nou repositori al sistema, corresponent a l'oficial de Falco, per tal de poder descarregar-se d'allà el paquet "falco". Concretament:

* A sistemes Ubuntu cal executar les següents comandes (la primera descarrega la clau associada al repositori propi de Falco, afegit a la segona comanda i actualitzat a la tercera, des del qual es descarregarà i instal·larà finalment -mitjançant la quarta comanda- el paquet "falco"):

curl -s https://falco.org/repo/falcosecurity-packages.asc | sudo gpg --dearmor -o /usr/share/keyrings/falco-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/falco-archive-keyring.gpg] https://download.falco.org/packages/deb stable main" | sudo tee -a /etc/apt/sources.list.d/falcosecurity.list
sudo apt update
sudo apt install dialog falco <-A les pantalles interactives que apareixen, cal contestar "Modern eBPF" en la 1a i "Yes" en la 2a

NOTA: La primera pantalla interactiva que apareix a l'hora d'instal·lar Falco demana el "driver" que volem que s'instal·li, que és el que farà servir Falco per recopilar tots els events del sistema detectats (bàsicament, les crides al sistema realitzades). Falco disposa de diferents "drivers" per fer aquesta tasca, com són el de tipus "Kmod" (consistent en la càrrega d'un mòdul del kernel "ad-hoc" per realitzar aquesta tasca de recolecció) o, el que hem triat, de tipus "Modern eBPF", un mecanisme molt menys intrusiu i robust. En tot cas, un cop triat el "driver", es posarà en marxa automàticament (i a cada reinici igual) el (únic) dimoni Systemd corresponent ("falco-kmod.service", "**falco-modern-bpf.service**", etc)
NOTA: La segona pantalla interactiva que apareix a l'hora d'instal·lar Falco pregunta si volem actualitzar periòdicament les regles del repositori oficial de regles proporcionades de forma online per Falco. Si diem que sí, això farà que el servei "**falcoctl-artifact-follow.service**" es posi en marxa automàticament cada cop que ho faci també el servei "falco-modern-bpf.service" -o el corresponent al "driver" indicat- i s'aturi automàticament també quan l'anterior es pari; aquest servei "falco-artifact-follow.service" és el responsable de buscar noves regles del repositori online de Falco i, si s'escau, actualitzar l'arxiu de regles local (mitjançant l'execució de una comanda d'administració interna de Falco anomenada *falcoctl*, que estudiarem aviat). El fet que la posada en marxa/aturada del servei "falcoctl-artifact-follow" depengui de la posada en marxa/aturada del servei "falco-modern-bpf" fa que, en general, no haguem de preocupar-nos per res de l'administració del primer

* A sistemes Fedora cal executar les següents comandes (la primera descarrega la clau associada al repositori propi de Falco, afegit a la segona comanda, des del qual es descarregarà i instal·larà -mitjançant la tercera comanda- el paquet "falco" i es posarà en marxa -mitjançant la quarta comanda- el servei corresponent):

sudo rpm --import https://falco.org/repo/falcosecurity-packages.asc
sudo curl -s -o /etc/yum.repos.d/falcosecurity.repo https://falco.org/repo/falcosecurity-rpm.repo
sudo dnf install falco
sudo systemctl --now enable falco-modern-bpf <-Automàticament es posarà en marxa també el servei "falcoctl-artifact-follow"

NOTA: Si no es vol que el servei "falcoctl-artifact-follow" es posi en marxa juntament amb "falco-modern-bpf", caldrà deshabilitar-lo permanentment amb la comanda *sudo systemctl mask falcoctl-artifact-follow* (per desfer aquesta deshabilitació la comanda adient seria *sudo systemctl unmask falcoctl-artifact-follow*). Aquesta nota es vàlida per qualsevol mètode d'instal·lació de Falco que s'hagi fet servir, en qualsevol distribució

En tot cas, un cop instal·lat Falco, i sense haver de definir cap regla personalitzada si no es necessita (ja que la instal·lació ja aporta el conjunt de regles oficials més actualitzat fins el moment), ja es pot tenir el sistema protegit si tenim en marxa el dimoni corresponent en segon pla (cosa que ja passa si seguim les instruccions d'instal·lació d'Ubuntu o Fedora indicades als paràgrafs anteriors; això ho podem comprovar fent simplement *systemctl status falco-modern-bpf*; en aquest sentit, recordem que es pot aturar amb *sudo systemctl stop falco-modern-bpf* o deshabilitar amb *sudo systemctl disable falco-modern-bpf*, a l'igual que es pot iniciar manualment i habilitar amb els verbs contraris, *start* i *enable*). També podem optar per executar Falco "manualment" en primer pla de forma puntual (si el dimoni està aturat!), simplement escrivint la comanda *sudo falco -o engine.kind=modern_ebpf*

NOTA: Tant el que fa "per sota" el dimoni "falcoctl-artifact-follow.service" com el dimoni "falco-modern-bpf" es pot conèixer si s'observa la sortida de la comanda *systemctl cat falcoctl-artifact-follow* o *systemctl cat falco-modern-bpf*, respectivament (en concret, el valor de la línia "ExecStart=". En el primer cas es pot veure, concretament, que a la pràctica s'acaba executant la comanda *falcoctl* (que estudiarem aviat) i en el segon cas es pot veure que s'acaba executant justament la comanda *falco -o engine.kind=modern_ebpf*

NOTA: El binari *falco* té diversos paràmetres que es poden indicar "ad-hoc" en el moment d'executar-lo. Alguns són:

-c /ruta/fitxer.config : Indica la ruta del seu fitxer de configuració (en lloc de l'arxiu per defecte, "/etc/falco/falco.yaml", del qual parlarem tot seguit)

-o opcio=valor : Estableix el valor indicat a l'opció de configuració indicada (sobreescrivint el possible valor que aquesta pugui tenir dins del fitxer de configuració). Aquest paràmetre es pot indicar més d'un cop.

-r /ruta/fitxer.regles : Indica la ruta del/s fitxer/s de regles (aquest paràmetre es pot indicar més d'un cop), enlloc dels arxius indicats en el seu arxiu de configuració

-V /ruta/fitxer.regles : Llegeix el contingut del fitxer de regles indicar per validar la seva sintaxis, i surt

-L : Mostra el nom i descripció de totes les regles, i surt

-M nº : Estableix el nombre de segons que romandrà falco funcionant abans de sortir (per defecte és infinit)

-S nº : Estableix el nombre de bytes del contingut transferit en crides al sistema de tipus I/O (com per exemple, *read()* o *write()*, entre altres) que Falco guardarà en el registre de l'event. Per defecte és 80; cal tenir en compte que si s'augmenta molt, pot haver penalització en el rendiment.

-A : Per rendiment, Falco no detecta totes les crides al sistema existents (són les que apareixen llistades amb el paràmetre *-i* o, el que és el mateix, les marcades com a "No" en la columna "Default" aquí: https://falco.org/docs/reference/rules/supported-events). El procediment que fa servir Falco per decidir quines són les crides detectades per defecte és relativament complex i s'explicarà més endavant. Per forçar, però, a que les detecti totes, cal afegir aquest paràmetre *-A*. Cal saber que si s'indica pot haver penalització en el rendiment.

Per defecte, Falco llegeix dos fitxers YAML per poder funcionar: "**/etc/falco/falco.yaml**" i "**/etc/falco_rules.yaml**". El primer fitxer serveix per configurar el binari "falco" en sí (amb diferents directives que defineixen el seu comportament, tal com veurem als exercicis) El segon fitxer serveix per definir les regles que per defecte s'utilitzaran per detectar events específics relacionats amb el comportament del sistema (i, en conseqüència, activar les alertes associades, si s'escaiés). Aquest darrer fitxer ja conté "de fàbrica" un conjunt predeterminat de regles ja dissenyades per proporcionar una bona cobertura en diverses situacions i la idea és que aquest fitxer de regles no es modifiqui ja que és reemplaçat automàticament a cada nova versió del programari que s'insta·li (o actualització que faci el servei "falcoctl-artifact-follow"). Per afegir/anul·lar/modificar regles personalitzades cal emprar, en canvi, el fitxer de regles "**/etc/falco/falco_rules.local.yaml**" (buit per defecte); aquest fitxer no es substituirà a cada nova versió del programa i es llegeix després de "falco_rules.yaml", de manera que les regles definides a "falco_rules.local.yaml" sempre tenen prioritat.

D'altra banda, cal saber que el projecte Falco ofereix altres conjunts de regles llestes per utilitzar les quals, tot i no venir incorporades "de sèrie" en el paquet Falco, es poden descarregar a banda i incorporar al conjunt de regles que Falco entendrà. La raó de per què aquestes altres regles semioficials no es distribueixen juntament amb el paquet "oficial" (tot i estar mantingudes en el repositiori oficial del projecte, veieu https://github.com/falcosecurity/rules i https://falcosecurity.github.io/rules) s'explica a continuació:

* D'una banda hi ha les regles anomenades "Incubating", les quals proporcionen un nivell similar d'estabilitat i robustesa que les regles oficials però són d'un abast més limitat i concret perquè es refereixen a casos d'ús més específics on estan involucrades aplicacions concretes que poden no ser rellevants per tots els usuaris. Es troben disponibles aquí: https://raw.githubusercontent.com/falcosecurity/rules/main/rules/falco-incubating_rules.yaml

* D'altra banda hi ha les regles anomenades "sandbox", les quals són més experimentals i, per tant, la seva utilitat i rellevància encara s'està avaluant. Es troben disponibles aquí: https://raw.githubusercontent.com/falcosecurity/rules/main/rules/falco-sandbox_rules.yaml

Per gaudir d'algun d'aquests conjunts de regles (l´"incubating" i/o el "sandbox") cal descarregar l'arxiu YAML corresponent (ja sigui mitjançant *curl* o *wget* apuntant a les URL indicades anteriors, o bé mitjançant la comanda específica *falcoctl* que estudiarem aviat) i guardar-lo sota la carpeta "/etc/falco". Un cop fet això, caldrà indicar a Falco que en faci ús d'aquests nous arxius (perquè si no els obviarà) editant el seu arxiu de configuració (és a dir, l'arxiu "/etc/falco/falco.yaml") i indicant-hi les rutes d'aquests nous arxius sota l'opció "rules_file". Finalment, caldrà reiniciar el servei un cop fet tots aquests passos. De totes formes, tot això ho detallarem pas a pas com fer-ho en els exercicis.

Les regles Falco predeterminades proporcionades pels fitxers YAML oficials (i les regles personalitzades escrites per nosaltres al fitxer "falco_rules.local.yaml") poden tenir un abast molt diferent, però totes segueixen el mateix patró de definició. A continuació es presenta un exemple de regla:

```
- rule: Modify binary dirs
  desc: An attempt to modify any file below a set of binary directories
  condition: (bin_dir_rename) and modify and not package_mgmt_procs and not exe_running_docker_save
  output : >
    File below known binary directory renamed/removed (user=%user.name command=%proc.cmdline
    parent_command=%proc.pcmdline operation=%evt.type file=%fd.name %evt.args)
  priority: ERROR
```

Com es pot veure, una regla consta de pocs elements obligatoris, els quals són:

rule: Identificador/nom de la regla
desc: Descripció breu de la regla
condition: Regla pròpiament dita, escrita usant una sintaxi particular que serveix per indicar els filtres desitjats per sel·leccionar només els events amb què estiguem interessats (sintaxi que s'explicarà en breu) i/o opcionalment, afegint-hi macros Falco (també explicades avall).
output: Missatge de sortida mostrat a *stdout* quan la regla s'apliqui (el "text de l'alerta")
priority: Nivell de prioritat de la regla (el seu valor pot ser "DEBUG", "INFORMATIONAL", "NOTICE", "WARNING", "ERROR", "CRITICAL", "ALERT" o "EMERGENCY").

> NOTA: Si el valor de la "priority" d'una regla és igual o més greu que el valor l'indicat a la línia *priority* de l'arxiu "falco.yaml", la regla associada es tindrà en compte; si no, la regla s'ignorarà. Com que per defecte la línia priority de l'arxiu "falco.yaml" val "debug", per defecte totes les regles són reconegudes i, per tant, la "priority" d'una regla simplement serveix per establir una certa jerarquia d'importància, res més.

A una regla es poden afegir a més altres elements opcionals (la llista completa dels elements possibles en una regla la podeu trobar a https://falco.org/docs/rules/basic-elements/#advanced-rule-syntax). Entre ells podem destacar:

enabled: false : Per defecte totes les regles estan habilitades a no ser que s'indiqui aquest element. D'aquesta manera, no caldrà esborrar-la o comentar-la per complet per a què Falco la ignori
tags: [unaetiqueta,unaaltra...] : Indica una llista d'etiquetes associades a la regla en qüestió. Les etiquetes permeten que Falco només tingui en compte un/s conjunt/s de regles concrets (l'etiquetat/s amb determinada/es etiqueta/es, justament) si a l'executable *falco* s'indica el paràmetre *-t unaetiqueta -t unaaltra ...* (o a l'inversa: tingui en compte totes les regles excepte un/s conjunt/s de regles concrets si al seu executable s'indica el paràmetre *-T unaetiqueta -t unaaltra ...*). Alternativament, també es pot indicar el paràmetre *-D cadena*, on "cadena" ha de ser una subcadena del nom de la/e regla/es que es vulgui/n ignorar.

> NOTA: Existeixen un conjunt d'etiquetes ja predefinides llestes per usar; es poden consultar a https://falco.org/docs/rules/controlling-rules/#tags-for-current-falco-ruleset

Una "macro" Falco és un nom assignat a una determinada condició. Serveixen com "dreceres" per poder indicar de forma senzilla condicions complexes i llargues en una o més regles. De fet, existeixen unes quantes macros predefinides als fitxers YAML oficials que poden ser molt útils a l'hora de definir les nostres pròpies regles. En tot cas, per exemple, a la regla anterior hi havia la macro "bin_dir_rename", la qual caldria haver-la definit anteriorment, així:

```
- macro: bind_dir_rename
  condition: >
     evt.arg[1] startswith /bin/   or  evt.arg[1] startswith /sbin/   or  evt.arg[1] startswith /usr/bin
```

La sintaxi de les condicions a definir en les regles o macros Falco per filtrar els events a detectar es basa en la comparació (mitjançant operadors com *=*, *!=*, *<*, *<=*, *>*, *>=*, *contains*, *icontains* -case-insensitive-, *in* , *exists*, *startswith* ...la llista completa és a https://falco.org/docs/rules/conditions/#operators) d'una determinada dada contra un determinat valor (o vàries dades amb sengles valors respectius, si les comparacions es concatenen mitjançant operadors com *and*, *or* i/o *not* i parèntesis).

NOTA: Per conèixer més la sintaxi de les condicions Falco al detall, consulteu https://falco.org/docs/rules/conditions

També existeix el concepte de "llista" Falco, la qual no és més que un nom assignat a un conjunt d'elements indicats (els quals poden ser, al seu torn altres llistes o macros). Existeixen també unes quantes llistes predefinides als fitxers YAML oficials. La seva sintaxis de la seva definició és:

```
- list: nomLlista
  items: [elementArray1,elementArray2,...]
```

És possible ampliar/substituir definicions de les regles, macros o llistes que s'hagin establert anteriorment en un mateix fitxer de regles (o en un altre fitxer de regles carregat abans de l'actual -l'ordre de càrrega ve donat per l'ordre en què s'indiquen a l'opció *rules_file* de l'arxiu "falco.yaml" o bé l'ordre en què s'indiquen amb el paràmetre *-r*). Per fer això, en aquest darrer arxiu cal definir una llista/macro/regla amb el mateix nom que el que es vol ampliar, el valor de l'element que es vol alterar (*items:* en cas d'una llista, *condition:* en cas d'una macro i *desc:*, *condition:*, *output:* i/o *tags:* en cas d'una regla) i llavors afegir una secció ***override:*** que ha de contenir l'element el nom de l'element alterat (*items:*, *condition:*, *output:*, etc) amb el valor ***append*** (si es vol afegir el valor nou a l'existent) o ***replace*** (si es vol substituir. Per exemple, si tenim definida una llista com...:

```
- list: my_programs
  items: [ls, cat, pwd]
```

...podríem ampliar aquesta llista amb un valor més simplement redefinint la llista així...:

```
- list: my_programs
  items: [cp]
  override:
     items: append
```

...o bé canviar la llista completament, així:

```
- list: my_programs
  items: [cp]
  override:
     items: replace
```

NOTA: En el cas de fer servir *append* en una línia *condition:* , la condició a afegir ha de començar obligatòriament amb un "*and* ..." o un "*or*..." per indicar sintàcticament així que s'està afegint aquest valor al valor precedent a la condició original
NOTA: En una sola regla es pot afegir/substituir de forma independent diferents elements (*condition:*, *output;* etc) dins de la secció *override:*

Les dades a comparar en una condició solen ser (encara que no només) aspectes concrets de crides al sistema concretes detectades en temps real. Algunes de les dades més comunes (les quals, per cert, també es poden indicar, sempre que es precedeixin amb el símbol "%", com a part del missatge de sortida definit en les regles de Falco per tal de mostrar-hi a *stdout* el seu valor concret detectat) poden ser les següents (la llista completa dels noms de les dades possibles es troba a https://falco.org/docs/reference/rules/supported-fields):

evt.type : nom de l'event (per exemple: *openat*, *read*, etc si es tracten de crides al sistema). La llista completa dels noms reconeguts es troba a https://falco.org/docs/reference/rules/supported-events
evt.args : cadena formada per tot els paràmetres de l'event (és a dir, de la crida) un darrera l'altre
evt.arg.nomParam : paràmetre concret (indicat pel seu nom) de l'event en qüestió
evt.arg[nº] : paràmetre concret (indicat per la seva posició) de l'event en qüestió
evt.time : "timestamp" de la crida al sistema ("event") en qüestió detectada
evt.dir : instant de detecció de l'event (pot valer ">", per indicar "a iniciar-se l'event" o "<" per indicar "en finalitzar l'event"; el més sovint és que ens interessi sobretot els events de finalització perquè és quan es podrà tenir més informació sobre què/com s'ha realitzat l'event en qüestió
evt.buffer : contingut del "buffer" si l'event ho proporciona (com són les crides *read*, *write*...)
proc.pid : PID del procés que ha generat l'event
proc.name : nom del procés que ha generat l'event. La ruta absoluta es pot obtenir de *proc.exepath*
proc.args : cadena formada per tot els paràmetres del procés que ha generat l'event
proc.cmdline : cadena formada per *%proc.name + %proc.args*
thread.tid : TID del thread que ha generat l'event (es correspon al PID en processos d'un sol thread)
user.uid : UID de l'usuari que ha generat l'event
user.name : nom de l'usuari que ha generat l'event
fd.num : nombre del "file descriptor" amb el qual se suposa que l'event en qüestió hi interacciona
fd.name : nom del "file descriptor" amb el qual se suposa que l'event en qüestió hi interacciona
fd.type : tipus de "file descriptor" (pot valer "file","directory", "ipv4","ipv6" -per connexions de xarxa, "unix" -pels sockets d'aquests tipus-, "pipe" -per canonades-, etc). En el cas dels "fd" de tipus "ipv4", podem utilitzar també les dades *fd.l4proto* (que pot valer "tcp", "udp" o "icmp"), *fd.sip* (que valdrà una adreça IP que representa que és la del servidor -és a dir, la de l'extrem d'una connexió que està a l'escolta via la implementació d'un socket "passiu"-), *fd.sport* (que valdrà un nombre de port del servidor), *fd.cip* (que valdrà una adreça IP que representa que és la del client -és a dir, la de l'extrem d'una connexió que fa la petició via la implementació d'un socket "actiu"-), *fd.cport* (que valdrà un nombre de port del client), *fd.sip.name* o *fd.cip.name* (noms DNS en lloc d'IPs), etc

> NOTA: Un "fd" és un identificador numèric que identifica de manera única un fitxer dins d'un procés. Això significa que podeu veure el mateix número "fd" utilitzat més d'una vegada, però haurà de ser en processos diferents. Seguint una combinació de procés/FD, es pot fer un seguiment de l'activitat d'E/S específica. És molt important tenir en compte que en un sistema Linux, un "fitxer" pot tenir moltes coses diferents: un fitxer normal, una connexió de xarxa (socket), una canonada, un temporitzador, un senyal, els fluxos stdout, stderr i stdin (en concret, recordeu que el fd=0 representa l'entrada estàndard -normalment el teclat-, el fd=1 representa la sortida estàndard -la pantalla- i el fd=2 la sortida d'error -normalment la pantalla també-), etc.

> NOTA: A Falco, els "file descriptors" es resolen per defecte. Això significa que, sempre que sigui possible, el número FD va seguit d'una representació llegible del propi FD (la tupla IP:port <-> IP:port per a connexions de xarxa, el nom dels fitxers, etc). El format exacte que s'utilitza per representar un FD és el següent: *num (<tipus> cadena_resolta)* on: *num* és el número FD; "cadena_resolta" és la representació resolta del FD (per exemple, "127.0.0.1:40370->127.0.0.1:80" per a un sòcol TCP) i el *tipus* és una codificació d'una sola lletra que indica el tipus de fd i que pot ser un dels següents: "f" per a fitxers, "4" per a sockets IPv4, "6" per a sockets IPv6, "u" per a sockets Unix, "s" per a senyals, "e" per a events," i "per fds inotify i "t " per temporitzadors

EXERCICIS:

0.-a) Instal.la Falco a una màquina virtual qualsevol (Ubuntu o Fedora) executant les comandes adients, indicades a la teoria.

> NOTA: La versió actual de Falco té un "bug" que fa que calgui, després d'instal·lar el paquet, crear a mà la carpeta "/usr/share/falco". És a dir, caldrà que executis la comanda *sudo mkdir /usr/share/falco* Si no ho fas, el servei d'actualitzacions automàtiques de regles "falcoctl-artifact-follow.service" no funcionarà. A més, cal afegir també el paràmetre *–no-verify* a la línia *ExecStart=* de l'arxiu "/usr/lib/systemd/system/falcoctl-artifact-follow.service" (i tot seguit executar *sudo systemctl daemon-reload* i reiniciar el servei)

b) Descarrega el conjunt de regles "incubating" i "sandbox" per tal que Falco les tingui en compte, a més de les regles oficials. Això ho pots fer executant la següent comanda: *sudo falcoctl artifact install falco-incubating-rules falco-sandbox-rules*

> NOTA: La comanda anterior automàticament descarrega del repositori oficial de Falco els arxius de regles indicats i els copia sota la carpeta "/etc/falco" del sistema. Per saber quins altres arxius de regles alternatius es poden descarregar (i des d'on) es pot executar la comanda *falcoctl artifact list --type rulesfile* En tot cas, per conèixer les diferents possibilitats que ofereix la comanda *falcoctl* en general, consulta aquest article: https://falco.org/blog/falcoctl-install-manage-rules-plugins

bII) Edita l'arxiu de configuració general de Falco ("/etc/falco/falco.yaml") per a què la seva secció *rules_file:* tingui el següent contingut (en negreta s'ha indicat les línies que has d'afegir) i tot seguit reinicia el servei "falco-modern-bpf":

> *rules_file:*
> *- /etc/falco/falco_rules.yaml*
> ***- /etc/falco/falco-incubating_rules.yaml***
> ***- /etc/falco/falco-sandbox_rules.yaml***
> *- /etc/falco/falco_rules.local.yaml*
> *- /etc/falco/rules.d*

bIII) En els comentaris sobre la línia *rules_file:* de l'arxiu "falco.yaml" hi apareix el següent paràgraf. Després de llegir-lo, digues quina funcionalitat creus que té llavors la carpeta */etc/falco/rules.d* (actualment buida): *"It's important to note that the files or directories are read in the order specified here (in rules_file: section). In addition, rules are loaded by Falco in the order they appear within each rule file."*

1.-Observa el contingut del fitxer de configuració general de Falco ("/etc/falco/falco.yaml") i respon:

a) ¿Per què, tot i que la secció *engine:* conté la línia *kind: kmod*, en realitat cada cop que s'inicia el servei "falco-modern-ebpf" s'està utilitzant un altre "engine" de captura d'events, l'anomenat "modern_ebpf"? Pista: observa el valor de la directiva *ExecStart=* mostrat per *systemctl cat falco-modern-bpf*

> NOTA: Cada valor possible de la línia *kind:* té al seu torn, com es pot veure, una subsecció amb el seu nom on es poden indicar certs paràmetres específics del seu funcionament particular

b) ¿Que implica que la línia *watch_config_files* tingui assignat el valor *true*? Pista: llegeix el comentari corresponent per saber la resposta

c) ¿Què implica que la línia *priority* tingui assignat el valor *debug* ? ¿Quina altres valors podria tenir assignats? Pista: llegeix el comentari corresponent per saber la resposta

d) ¿Quina diferència hi hauria entre establir el valor *true* o *false* a la línia *json_output* ? I en el cas de què valgués *true*, ¿per a què serviria llavors establir a *true* la línia *json_include_output_property* ? Pista: llegeix el comentari corresponent per saber la resposta

e) ¿Què implica que les seccions *stdout_output:* i *syslog_output:* continguin ambdues la línia *enabled: true* ?

f) ¿Què implicaria que la secció *file_output:* contingués la línia *enabled: true* ? ¿Quina diferència hi hauria llavors entre establir el valor *true* o *false* en la seva línia *keep_alive*? Pista: llegeix el comentari corresponent per saber la resposta

g) ¿Què implicaria que la secció *http_output:* contingués la línia *enabled: true* ? ¿I què indicaria llavors la línia *url* ? ¿I les línies *insecure: false* més *ca_cert: "ca.crt"* ? Pista: llegeix el comentari corresponent per saber la resposta

h) ¿Què implicaria que la secció *program_output:* contingués la línia *enabled: true* ? ¿I què indicaria llavors la línia *program* ? ¿Quina diferència hi hauria entre establir el valor *true* o *false* a la línia *keep_alive*? Pista: llegeix el comentari corresponent per saber la resposta

i) ¿Què implica que les línies *log_stderr* i *log_syslog* valguin *true* ? ¿Què implica que la línia *log_level* tingui assignat el valor *info* ? ¿Quina altres valors podria tenir assignats? Pista: llegeix el comentari corresponent per saber la resposta

> NOTA: Teniu més informació sobre com configurar Falco a https://falco.org/docs/reference/daemon/config-options

2.-Després d'assegurar-te que Falco no s'estigui executant en segon pla (fes *sudo systemctl status falco-modern-bpf* per comprovar-ho i, si calgués, *sudo systemctl stop falco-modern-bpf*), executa'l en primer pla (així: *sudo falco -o engine.kind=modern_ebpf*). A partir d'aquí:

a) En un altre terminal diferent, prova de modificar un fitxer que estigui dins de la carpeta "/etc" (per exemple, modifica algun comentari del fitxer "/etc/fstab"). ¿Què veus al terminal on s'està executant Falco?

aII) ¿I si ara crees un nou fitxer dins de la carpeta "/bin" (per exemple, fent *sudo touch /bin/virus*), ¿què veus al terminal on s'està executant Falco? ¿I si executes la comanda *sudo cat /etc/shadow* , què veus?

b) Atura l'execució de Falco amb CTRL+C ¿Quins missatges veus si executes la comanda *journalctl -e _COMM=falco*? ¿Degut a l'activació de quina línia/secció de configuració vista a l'exercici anterior podem veure aquests missatges?

> NOTA: Existeix un programa, pertanyent al projecte Falco oficial, que serveix per generar un determinat conjunt d'events sospitosos (i reals, compte!) per tal de comprovar la detecció realitzada per una instància Falco que estigui en marxa. És https://github.com/falcosecurity/event-generator , però no el farem servir

3.-a) Digues per a què serviria el següent contingut dins del fitxer "falco_rules.local.yaml" i escriu-lo:

```
- rule: pepita
  desc: adescobrir
  condition: (proc.name=cat or proc.name=nano) and evt.dir=<
  output: misteri misteri (usuari=%user.name comanda=%proc.cmdline syscall=%evt.type fitxer=%fd.name)
  priority: INFO
```

aII) Inicia el servei Falco (*sudo systemctl start falco*) i provoca alguna acció relacionada amb la regla anterior per tal de veure aparèixer, amb la comanda *journalctl -u falco -f*, els missatges corresponents en el registre del sistema (estem suposant que la línia syslog_output està "enabled")

b) Digues per a què serviria el següent contingut dins del fitxer "falco_rules.local.yaml" i escriu-lo:

```
- rule: pepita2
  desc: adescobrirtambé
  condition: ((proc.name=cat and evt.type=openat) or (proc.name=sshd and evt.type=accept)) and evt.dir=<
  output: temps=%evt.time syscall=%evt.type arguments=%evt.args fdtipus=%fd.type
  priority: INFO
```

bII) Reinicia el servei Falco (*sudo systemctl restart falco*) i provoca alguna acció relacionada amb la regla anterior per tal de veure aparèixer els missatges corresponents al registre del sistema

c) Digues per a què serviria el següent contingut dins del fitxer "falco_rules.local.yaml" i escriu-lo:

```
- rule: pepita3
  desc: tambéadescobrir
  condition: evt.type=chdir and evt.dir=<
  output: Ara %proc.name es mou a %evt.arg.path
  priority: INFO
```

cII) Reinicia el servei Falco i provoca alguna acció relacionada amb la regla anterior per tal de veure aparèixer els missatges corresponents al registre del sistema

d) Digues per a què serviria el següent contingut dins del fitxer "falco_rules.local.yaml" i escriu-lo:

```
- list: unallistadeprogrames
  items: [ls,cat,pwd]
- macro: uneventconcret
  condition: evt.type=openat and evt.dir=<
- rule: pepita4
  desc: mesadescobrirencara
  condition: proc.name in (unallistadeprogrames) and uneventconcret
  output: A veure què surt (usuari=%user.name comanda=%proc.cmdline fitxer=%fd.name)
  priority: INFO
```

dII) Reinicia el servei Falco i provoca alguna acció relacionada amb la regla anterior per tal de veure aparèixer els missatges corresponents al registre del sistema

e) Digues per a què serviria el següent contingut dins del fitxer "falco_rules.local.yaml" i escriu-lo:

```
- macro: unaltreevent
  condition: fd.type=ipv4 and fd.l4proto=tcp and fd.sport=22 and evt.dir=<
- rule: pepita5
  desc: mesencara
  condition: unaltreevent
  output: clientIP=%fd.cip usuari=%user.name comanda=%proc.cmdline syscall=%evt.type
  priority: INFO
```

eII) Reinicia el servei Falco i provoca alguna acció relacionada amb la regla anterior per tal de veure aparèixer els missatges corresponents al registre del sistema

f) ¿Quin tipus d'esdeveniment intentarien detectar cadascuna de les següents condicions (associades cadascuna a una hipotètica regla, que podria tenir com a sortida valors interessants com *%user.name*, *%proc.cmdline*, *%fd.name*, etc)? Digues també alguna/es comanda/es que hauries d'executar per a què Falco generés l'alerta pertinent:

proc.name=cat and evt.type=write and fd.num=1 and evt.dir=<
proc.name=cat and evt.type=write and fd.name contains /etc and evt.dir=<

NOTA IMPORTANT: Si es volen provar les condicions on s'indica la crida "write", caldrà prèviament afegir el paràmetre ***-A*** a l'executable de *falco* (en el cas de fer-lo servir com a dimoni, aquest haurà d'estar indicat a la línia *ExecStart=* de l'arxiu "/usr/lib/systemd/system/falco-modern-bpf.service") perquè aquesta és una de les crides que per defecte Falco no les detecta (per una qüestió de rendiment)

NOTA: Si es vol obtenir el contingut concret escrit per la crida "write" (o altres, o llegit per la crida "read" o altres) a la sortida es pot indicar la dada %evt.buffer. No obstant, per defecte Falco té limitada la mida d'aquest valor a només 80 bytes (per una qüestió de rendiment). Si es vol observar més informació, caldrà prèviament afegir el paràmetre ***-S nºbytesmax*** a l'executable de *falco* (d'igual forma que s'ha comentat a la "NOTA" anterior)

g) ¿Quin tipus d'esdeveniment intentarien detectar cadascuna de les següents condicions (associades cadascuna a una hipotètica regla? Digues també alguna/es comanda/es que hauries d'executar per a què Falco generés l'alerta pertinent:

evt.type=execve and evt.arg.ptid=bash and evt.dir=<
evt.type=execve and proc.name=rm and evt.res=SUCCESS and evt.dir=<

NOTA: Recorda que la crida "execve" és la 1ª crida feta per tot programa per començar-se a "autoexecutar" D'altra banda, el paràmetre "ptid" d'aquesta crida indica el PID del pare del programa que ha fet aquesta crida; per tant, en aquest cas s'està monitoritzant tots els procesos generats des d'un terminal Bash.
NOTA: La dada *evt.res* indica el valor de retorn de la crida en qüestió; en el cas d'una execució correcta, aquest valor sol ser 0, tot i que no sempre; afortunadament, Falco ofereix la possibilitat d'indicar la cadena "SUCCESS" com a sinònim d'execució correcta, sigui el valor concret que sigui el retornat. D'altra banda, també hi ha la possiblitat d'indicar, enlloc del valor numèric concret adient, alguna cadena específica per errors coneguts (com per exemple per l'error "ENOENT", que apareix quan la crida en qüestió intenta obrir un fitxer o carpeta i no la troba -perquè no existeix o no hi és allà on se li ha dit, etc-, o "EPERM", que apareix quan hi ha un problema de permisos,...entre molts d'altres més)

gII) Digues per a què serviria la següent regla dins del fitxer "falco_rules.local.yaml" i escriu-la. Tot seguit, reinicia el servei Falco i provoca alguna acció relacionada amb aquesta regla per tal de veure aparèixer els missatges corresponents al registre del sistema

- rule: pepita6
desc: impossible
condition: evt.type in (execve,execveat) and not proc.name in (bash,sh,ls,rm) and evt.dir=<
output: Procés inesperat=> Nom: %proc.name Args: %proc.cmdline Pare: %proc.pname Ruta: %proc.cwd
priority: INFO

4.-a) Observa el contingut i comentaris del fitxer "falco_rules.yaml" (amb *less* et serà més fàcil) i respon:

*¿Què representa la macro "etc_dir"? ¿I la macro "open_write"?
*¿Què representen la llistes "passwd_binaries" i "shadowutils_binaries?" ¿Què indica la canonada de comandes indicada en el comentari que hi ha just a sobre de la definició d'aquestes llistes?

aII) Observa el contingut i comentaris del fitxer "falco-sandbox_rules.yaml" i respon:

*¿Què representa la macro "write_etc_common" (on s'aprofiten les macros anteriors: "etc_dir", "open_write" i "passwd_binaries"/"shadowutils_binaries")?
*¿Quin missatge mostra la regla "Write below etc" (on s'usa la macro anterior) i quan s'activa?

b) Observa el contingut i comentaris del fitxer "falco-sandbox_rules.yaml" i respon:

*¿Què representa la macro "rename"? ¿I la macro "remove"? ¿I la macro "modify"?
*¿Què representen les macros "bin_dir_rename" i "bin_dir_mkdir"?
*¿Quin missatge mostra la regla "Modify binary dirs" (on s'usen les macros anteriors)i quan s'activa?

c) Observa el contingut i comentaris del fitxer "falco-sandbox_rules.yaml" i respon:

*¿Què representa la macro "bin_dir"?
*¿Què representa la macro "package_mgmt_procs"?
*¿Quin missatge mostra la regla "Write below binary dir" (on s'usen les macros anteriors) i quan s'activa?

d) Observa el contingut i comentaris del fitxer "falco_rules.yaml" i respon:

*¿Què representa la llista "sensitive_file_names" i la macro "sensitive_files" (on s'usa aquesta llista)?
*¿En quines dues úniques regles s'utilitza la llista "sensitive_file_names"? ¿Quins missatges mostren aquestes dues regles i quan s'activen?

*¿Quin missatge mostra la regla "Read sensitive file untrusted" (on s'usa la macro "sensitive_files") i quan s'activa?

e) Observa el contingut i comentaris del fitxer "falco_rules.yaml" i respon:

*¿Què representa la macro "outbound"?
*¿Què representa la macro "ssh_non_standard_ports_network", basada en la llista "ssh_non_standard_ports"?
*¿Quin missatge mostra la regla "Disallowed SSH Connection Non Standard Port" (on s'usen les macros anteriors) i quan s'activa?

f) Observa el contingut i comentaris del fitxer "falco-sandbox_rules.yaml" i respon:

*¿Què representa la macro "inbound"? ¿I la macro "inbound_outbound"?
*¿Què representen les llistes "allowed_inbound_source_ipaddrs" i "allowed_inbound_source_networks"?
*¿Quin missatge mostra la regla "Unexpected inbound connection source" (on s'usen les macros i llistes anteriors) i quan s'activa?

g) Observa el contingut i comentaris del fitxer "falco-incubating_rules.yaml" i respon:

*¿Quin missatge mostra la regla "System procs network activity" i quan s'activa?
*¿Quin missatge mostra la regla "Create files below dev" i quan s'activa?

NOTA: Trobareu més exemples a https://sysdig.com/blog/sending-little-bobby-tables-detention i també a https://falco.org/docs/reference/rules/examples

5.-a) Modifica el contingut de l'arxiu "falco.yaml" per tal de què envïi (via *netcat*) la seva sortida, en format JSON, a un servidor remot (que pot ser la teva màquina real, per exemple) escoltant al port 1234. O dit d'una altra manera: estableix les següents línies:

```
json_output: true
program_output:
  enabled: true
  keep_alive: false
  program: nc x.x.x.x 1234
```

Reinicia el servei Falco i provoca de nou alguna acció relacionada amb la regla "pepita3" definida a l'exercici nº3. ¿Què veus a la pantalla del servidor Netcat?

NOTA: L'objecte JSON que treu Falco té els següents camps: "**time**" (el moment quan s'ha emés l'alerta, en format ISO8601), "**rule**" (el nom de la regla que ha causat l'alerta), "**priority**" (la prioritat d'aquesta regla), "**output**" (la sortida de la regla) i "**output_fields**" (tots els noms i valors dels diferents filtres utilitzats a la sortida de la regla)

b) Vés, amb un navegador qualsevol, a https://public.requestbin.com/r. Veuràs que aquest enllaç et genera dinàmicament un servidor propi amb un nom similar a https://xxxx.y.pipedream.net (on "xxxx.y" és una cadena aleatòria). Canvia ara la línia "program" de la configuració de Falco per a què ara, en lloc de fer servir el client Netcat, executi aquesta comanda: *curl -X POST -H "Content-Type:application/json" -d @- https://xxxx.y.pipedream.net* i reinicia el servei Falco. Finalment, provoca de nou alguna acció relacionada amb la regla "pepita3" definida a l'exercici nº3. ¿Què veus a la part esquerra de la pàgina https://xxxx.y.pipedream.net, on es mostren les dades que aquesta pàgina va rebent?

NOTA: Estem suposant que mantenim la línia *json_output* a *true* ; és per això que cal la presència del paràmetre -H anterior
NOTA: Recorda que si el valor del paràmetre *-d* de curl comença amb la lletra @, vol dir que la resta hauria de ser un nom de fitxer del qual llegir-ne les dades o "-" si es vol (com és el cas) que s'enviïn les dades rebudes per stdin
NOTA: Recorda que si la línia "keep_alive" del fitxer "falco.yaml" és *true*, la connexió de xarxa del programa (i el programa en si) es mantindrà funcionant per sempre; si és *false* (valor per defecte), la connexió de xarxa del programa (i el propi programa) es tancarà un cop s'enviï el flux de dades.
NOTA: Una alternativa a fer servir *program_output* en aquest cas concret seria fer servir la directiva *http_output*

c) Llegeix la pàgina oficial del projecte "Falcosidekick" (https://github.com/falcosecurity/falcosidekick) per esbrinar per a què serveix el servei homònim i, concretament aquest apartat (https://github.com/falcosecurity/falcosidekick?tab=readme-ov-file#with-falcoyaml) per saber quina és la configuració que hauria/en de tenir el servei "falco" per a poder fer-ne ús. Escriu el que has esbrinat.