Strace i Ltrace

Strace:

La comanda *strace* serveix per conèixer quines "crides al sistema" (es a dir, crides a funcions proporcionades pel kernel com ara la d'obrir, llegir, escriure o tancar fitxers o sockets, per exemple) fa servir un determinat programa d'usuari. Per tant, t'eina *strace* ens pot ajudar a esbrinar què fa realment una determinada comanda: quins fitxers llegeix, quins vol escriure, quins ports obre, a quins ports es vol connectar, etc, etc. És com ser un "espia" de tot allò que el programa fa a baix nivell.

La sintaxi general de strace és: ***strace comanda parametres***. on la comanda indicada s'executarà tal qual. Per exemple: *strace ls -l* executarà *ls -l* i, a mesura que avanci aquesta execució, anirà detectant i mostrant per pantalla (a stderr més en concret) totes les crides al sistema que realitzi *ls -l* La manera de mostrar-les és indicant, per cada crida detectada, primer el seu nom, després, entre parèntesis, els paràmetres concrets que aquesta crida ha rebut i, finalment, després d'un símbol "=", el valor retornat per aquesta crida (el significat del qual dependrà de la crida en qüestió: un valor igual a 0 sol significar "tot ok" i un valor diferent de 0 sol significar algun tipus d'error, encara que no sempre és així).

Com que les crides mostrades poden ser moltes depenent de la comanda inspeccionada, se sol afegir el paràmetre ***-e crida1,crida2,..*** per mostrar només les crides que ens interessen.Algunes crides comunes són:

open/openat : Obre un fitxer. Retorna un identificador numèric del fitxer anomenat "file descriptor" (fd)
access : Comprova els permisos d'un fitxer. Una crida més general seria "fstat" o també "statfs"
read : Llegeix un fitxer. Es pot filtrar exactament per quin fitxer si s'escriu read=nºfd
write : Escriu en un fitxer. Es pot filtrar exactament per quin fitxer si s'escriu write=nºfd
NOTA: El fd nº 1 representa stdout, el nº2 representa stderr i el nº0 representa stdin (el teclat)
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. També està "fchown" (que canvia el propietari)
close : Tanca un fitxer
unlink : Esborra un fitxer
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 : Executa un programa. Sol ser la primera crida que apareix
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

NOTA: Per saber més sobre les crides al sistema disponibles recomano la lectura de *man syscalls*.

Altres valors que poden acompanyar al paràmetre -e i que són més genèrics són:

file : Representa qualsevol crida al sistema que tingui un fitxer com a paràmetre
process : Representa qualsevol crida al sistema relacionada amb la gestió de processos
memory : Representa qualsevol crida al sistema relacionada amb la gestió de memòria
network : Representa qualsevol crida al sistema relacionada amb la xarxa
signal : Representa qualsevol crida al sistema relacionada amb la gestió de senyals
ipc : Representa qualsevol crida al sistema relacionada amb mecanismes IPC
desc : Representa qualsevol crida al sistema relacionada amb descriptors de fitxers (read,write...)

Altres paràmetres de *strace* són:

-o nomFitxer : Guarda la sortida de *strace* en el fitxer indicat (en comptes de per stderr)
-C: Mostra estadístiques al final de la sortida habitual: nº de crides,temps consumit per crida, errors...
-c : Mostra només les estadístiques. Es pot combinar amb *-S {time|calls| name}* per ordenar la sortida

-p PID1, PID2 : Aplica *strace* a comandes que ja estiguin en marxa en temps real
-P /ruta/fitxer : Monitoritza (només) totes les crides que pateix /ruta/fitxer per un determinat procés
-t : Mostra també l'hora i dia en la que s'ha realitzat cada crida
-tt : Similar a l'anterior però incloent els microsegons
-ttt : Similar a l'anterior però mostrant el temps amb format "UNIX Epoch"
-r : Mostra també el temps relatiu entre crida i crida
-T : Mostra també el temps que triga cada crida (la diferència entre l'hora d'inici i la de final)
-s nº : quantitat de caràcters que se mostren/guarden per cada crida (per defecte 32)
-f : Mostra també les crides realitzades pels possibles processos fills que apareguin
-ff : En el cas d'usar -o nomFitxer, es guarden les crides de cada procés fill en un fitxer diferent (anomenat "nomFitxer.pid")

Ltrace:

Cal distingir entre "crides al sistema" i "crides a funcions estàndar". Les primeres, tal com hem dit, són crides a funcions C proporcionades directament pel kernel; les segones també són crides a funcions C però en aquest cas aquestes estan proporcionades per la llibreria estàndar utilitzada a l'hora de compilar el programa en qüestió, la qual sol ser la GNU Glibc (https://www.gnu.org/software/libc) . Cada funció de la llibreria estàndar sol actuar -encara que no sempre- com a "wrapper" (embolcalls) d'una (o d'un conjunt de) crida/es al sistema, perquè, en general, les primeres realitzen tasques de baix nivell d'una manera menys directa (i per tant, més controlada, segura i portable) que les segones, estalviant així al desenvolupador haver de fer ell mateix les crides al sistema directament (sempre més delicades i perilloses). Per exemple, l'execució de la funció *printf("Hola mundo");* ,proporcionada per la llibreria estàndar, acaba provocant una crida a una funció proporcionada pel kernel anomenada *write(1,"Hola mundo",10);* -on el nº1 representa el descriptor de la sortida estàndar, stdout, i el nº10 és la quantitat de caràcters a escriure-. En tot cas, ja sigui perquè un programa cridi directament a una funció proporcionada pel kernel o bé perquè executi una funció proporcionada per la llibreria estàndar que realitza pel seu compte una crida a a una funció proporcionada pel kernel, *strace* registra i mostra totes aquestes crides sense distinció.

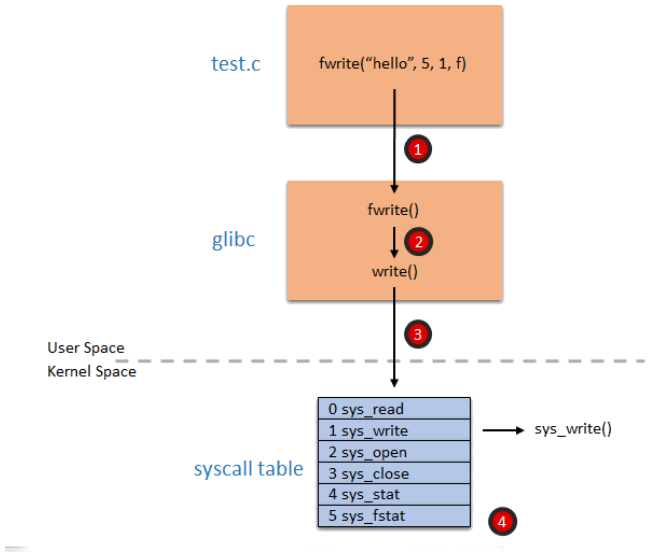
NOTA: No totes les funcions estàndar són"wrappers" de funcions del kernel. Per exemple, la funció strlen(), que serveix per conèixer la longitud d'una cadena passada com a paràmetre, no executa cap codi del kernel perquè no li cal.
NOTA: Per saber més sobre la llibreria estàndar, recomano la lectura de *man libc*

No obstant, si volem inspeccionar la crida a funcions incloses dins de (només) la llibreria estàndar, podem utilitzar una altra eina anomenada *ltrace* . Aquesta eina també admet el paràmetre -e, però en aquest cas, els valors que pot tenir són els noms de les funcions estàndars a inspeccionar (per exemple, **printf, malloc, free, strcmp, strlen, strcpy, getenv, opendir, readdir, closedir**, etc) i a més, concatenades mitjançant el símbol "+" en comptes de la coma. Per exemple, així: ***ltrace -e opendir+readdir+closedir ls***. Altres paràmetres de *ltrace* són:

-o nomFitxer : Guarda la sortida de *strace* en el fitxer indicat (en comptes de per stderr)
-c : Mostra les estadístiques de quantes crides s'han fet a cada funció i quant de temps en total s'ha gastat per cadascuna
-p PID1 : Serveix per aplicar l*trace* a comandes que ja estiguin en marxa en temps real
-t : Mostra també l'hora i dia en la que s'ha realitzat cada crida
-tt : Similar a l'anterior però incloent els microsegons
-ttt : Similar a l'anterior però mostrant el temps amb format "UNIX Epoch"
-r : Mostra també el temps relatiu entre crida i crida
-T : Mostra també el temps que triga cada crida (la diferència entre l'hora d'inici i la de final)
-S : Mostra també les crides al sistema (marcades amb "SYS") dins les crides a la llibreria estàndar

Cal tenir en compte, no obstant, que aquesta comanda només funciona per inspecciona executables de tipus ELF (els nadius de Linux). Per saber si un determinat programa és ELF o no es pot utilitzar la comanda *file /ruta/programa*

A continuació es mostra un simple diagrama (extret de https://sysdig.com/blog/fascinating-world-linux-system-calls/) on es mostra gràficament el que s'acaba d'explicar:



Llibreries:

Per saber si un determinat programa està enllaçat o no a la llibreria estàndar i, en general, per saber a quines altres llibreries hi està enllaçat, es pot fer ***ldd /ruta/programa*** . Molt sovint la comanda *ldd* mostrarà alguna de les següents llibreries (perquè són molt utilitzades per multitud de programes) que cal conèixer:

*"/lib/libc.so.X" (on X és un número): la llibreria estàndar GNU LibC pròpiament dita.

*"/lib/linux-vdso.so" : conté al seu interior un conjunt sel.leccionat de crides al sistema que són "injectades" pel kernel automàticament a la memòria del procés que carrega aquesta llibreria. Aquestes crides sel.leccionades tenen la particularitat d'executar-se en el context d'usuari i no pas de kernel (trobareu més informació a http://man7.org/linux/man-pages/man7/vdso.7.html o https://en.wikipedia.org/wiki/VDSO) fent així que la seva crida sigui més ràpida en no haver de canviar de context per executar-se.

*"/lib/ld.so" o "/lib/ld-linux.so" : carregador de llibreries del sistema (l'anomenat "loader"). Aquesta llibreria és ella mateixa carregada sempre automàticament el kernel abans de què s'executi el propi codi del programa en qüestió. La seva funcionalitat és trobar ràpidament les llibreries (inclosa l'estàndar GNU) que el programa en qüestió necessita (és a dir, les llibreries a les quals està enllaçat) per procedir seguidament a la seva correcta execució. Per a què aquest "loader" pugui trobar les llibreries necessàries en cada cas utilitza un "mapa", que és el fitxer "/etc/ld.so.conf", el qual consisteix simplement en una llista de rutes -una per línia- on haurà d'anar a buscar aquestes llibreries.

NOTA: No obstant, per a què aquest "mapa" sigui realment efectiu cal "compilar-lo" en el "mapa" que realment s'utilitza a la pràctica. Per fer aquesta compilació (necessària cada vegada que el contingut de "/etc/ls.so.conf" canvïi) cal executar la comanda ***ldconfig*** . El resultat de la "compilació" és un altre fitxer anomenat "/etc/ls.so.cache", el qual és un fitxer de text que conté les rutes exactes -una rera l'altra - de totes les llibreries que el "loader" reconeixerà.

NOTA: La comanda *ldconfig* se sol executar automàticament quan s'instal.la al sistema alguna llibreria nova però com a paràmetre més interessant per executar-lo manualment té el paràmetre -p, el qual serveix per mostrar la llista de llibreries carregades en aquest moment concret d'entre totes les que hi hagi llistades a /etc/ld.so.cache.

NOTA: Existeix la possibilitat d'indicar rutes complementàries on el "loader" buscarà llibreries sense haver de modificar l'arxiu "/etc/ld.so.conf" si s'indiquen (separades per ":") com a valor de la variable d'entorn LD_LIBRARY_PATH. Les rutes indicades allà tenen preferència sobre les indicades a "/etc/ld.so.conf", i això és utilitzat moltes vegades per "substituir" alguna llibreria del sistema per una llibreria pròpia equivalent (ubicada en un altre lloc). No obstant, el valor d'aquesta variable romandrà només fins que la màquina es reinicïi.

Sysdig i Falco

El programa Sysdig (https://www.sysdig.org) és un programa similar a *strace* però molt més versàtil i complet. No obstant, cal tenir en compte que fa servir un mòdul del kernel (anomenat "sysdig_probe" i responsable, entre altres coses, de generar el dispositiu de captura especial /dev/sysdig0), el qual no està integrat (encara) dins d'ell.

Si Sysdig s'executa (com a root) simplement "tal qual", així, *sysdig* , es mostrarà per pantalla (concretament, a "stdout") tots els events (els quals solen ser crides al sistema, encara que no només) generats en temps real. Cada event és mostrat en una línia diferent amb el següent format

%evt.num %evt.time %evt.cpu %proc.name (%thread.tid) %evt.dir %evt.type %evt.args

on:

evt.num is an incremental event number
evt.time is the event timestamp
evt.cpu is the CPU number where the event was captured
proc.name is the name of the process that generated the event
thread.tid is the TID that generated the event, which corresponds to the PID for single thread processes
evt.dir is the event direction, > for enter events and < for exit events
evt.type is the name of the event, e.g. 'open' or 'read'
evt.args is the list of event arguments. In case of system calls, these correspond to the system call's ones.

NOTA: An interesting argument which some system call have is the "file descriptor" (fd). A "fd" is a numeric ID that uniquely identifies a file inside a process. This means that you can see the same "fd" number used more than once, but it will have to be in different processes. By following a process/FD combination, you can track specific I/O activity. It's very important to note that in a Linux system, a "file" can be many different things: a regular file, a network connection (socket), the stdout, stderr and stdin streams, a pipe, a timer, a signal, etc

In Sysdig file descriptors are resolved by default. This means that, whenever possible, the FD number is followed by a human-readable representation of the FD itself: the tuple for network connections, the name for files, and so on. The exact format used to render an FD is the following: num(<type>resolved_string) where: *num* is the FD number; *resolved_string* is the resolved representation of the FD (e.g. 127.0.0.1:40370->127.0.0.1:80 for a TCP socket) and *type* is a single-letter-encoding of the fd type, and can be one of the following: "f" for files, "4" for IPv4 sockets, "6" for IPv6 sockets, "u" for unix sockets,"s" for signal Fds, "e" for event FDs, "i" for inotify FDs and "t" for timer FDs

NOTA: For most system calls, sysdig shows two separate entries: an enter one (marked with a '>') and an exit one (marked with a '<'). This makes it easier to follow the output in multi-process environments.

Alguns dels paràmetres més comuns per utilitzar amb sysdig són:

-w /ruta/fitxer : Guarda la sortida en el fitxer indicat (en comptes de per stdout)
-C nºMB : Parteix el fitxer de captura en fitxerets diferents amb un tamany cadascú dels MB indicats
-G nºs : Parteix el fitxer de captura en fitxerets diferents cada número de segons indicat
-e nº : Parteix el fitxer de captura en fitxerets diferents cada número d'events detectats
-W nº : Limita el número de fitxerets petits creats amb -C, -G o -e, realitzant una rotació de logs.
-n nº : Limita el número d'events capturats a la quantitat indicada
-r /ruta/fitxer : Llegeix (i mostra per pantalla) les captures guardades prèviament al fitxer indicat

Els filtres s'han d'escriure al final de la invocació de *sysdig* , com a últims paràmetres. Els filtres consisteixen simplement en indicar el nom del camp i el valor concret d'aquest que es vol observar. Per exemple, per veure l'activitat d'una determinada comanda (que pot estar ja funcionant o bé encara no) cal fer:

sysdig proc.name=cat

Als filtres es poden usar els següents operadors de comparació : =, !=, <, <=, >, >=, "contains", "icontains" (case-insensitive), "in" i "exists" . També es poden combinar entre ells mitjançant operadors booleans com: "and", "or" i "not" (i parèntesis). Per exemple, la següent comanda captura l'activitat de dos programes a la vegada, *cat* i *vi*:

sysdig proc.name=cat or proc.name=vi

El següent exemple mostra tots els fitxers que són oberts per programes que no siguin el *cat:*

sysdig proc.name!=cat and evt.type=open

El següent exemple mostra totes les connexions de xarxa rebudes per processos diferents de Apache:

sysdig proc.name!=apache and evt.type=accept

El següent exemple mostra totes les crides "execve" només si les realitza el procés pare "bash":

sysdig evt.arg.ptid=bash and evt.type=execve

NOTA: La diferència entre el filtre *evt.arg* i el filtre *evt.rawarg* és que el segon no fa resolució de PIDs, fds o codis d'error, deixant el valor en la seva forma numèrica "pelada" . Per exemple, es podria usar el filtre *evt.arg.res=ENOENT* per observar un codi de retorn d'error I/O específic o bé, ja que els codis d'errors són números negatius, es podria fer alternativament *evt.rawarg.res < 0 or evt.rawarg.fd < 0*

Per conèixer tots els filtres possibles, es poden llistar fent ***sysdig -l***

Per conèixer tots els events reconeguts, es poden llistar fent ***sysdig -L*** . En general, 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. Una llista molt més completa es troba a https://github.com/gregose/syscall-table .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, no hi ha res millor que consultar les pàgines del manual (concretament la secció 2, així: *man 2 read*, pe exemple).

NOTA: La secció 3 del manual està reservada per a les funcions de la llibreria estàndar. Es poden veure les diferents seccions del manual fent *man man*

Es pot personalitzar la sortida de *sysdig* es pot usar el paràmetre **-p** , el qual té una sintaxis similar a la de la funció printf() de C. Per exemple:

sysdig -p"*Usuari:%user.name Carpeta on s'entra:%evt.arg.path" evt.type=chdir

NOTA: L'asterisc del principi de la cadena serveix per indicar a Sysdig que imprimeixi la frase igualment encara que l'event no aporti els valors de tots els camps indicats a la frase (per defecte Sysdig no ho fa)

La personalització de la sortida es pot combinar amb els filtres que ja coneixem. Per exemple, la comanda següent mostra la sortida estàndar del procés cat (el paràmetre **-A** serveix per mostrar només la informació "humanament" llegible i no el contingut binari a pèl, i el paràmetre **-s** serveix per indicar que es volen capturar de cada crida a write() més dels usuals 80 bytes que Sysdig s'autoimposa)

sysdig -A -s 65000 -p"%evt.buffer" proc.name=cat and evt.type=write and fd.num=1

Un altre cas: podríem adaptar un exemple vist en línies anteriors per fer-ho més complet:

sysdig -p"%user.name) %proc.name %proc.args" evt.type=execve and evt.arg.ptid=bash

Per veure qui i amb quin programa està escrivint als fitxers ubicats dins de /etc podríem fer:

sysdig -p"User:%user.name Prog:%proc.name Fitx:%fd.name" evt.type=write and fd.name contains /etc

Igualment, per veure les connexions acceptades (Ip:port origen <-> Ip:port destí) per Apache...:

sysdig -p"%fd.name" proc.name=apache and evt.type=accept

A banda dels filtres, una altra funcionalitat molt interessant de Sysdig són els seus (anomenats per ells) "chisels". Els "chisels" són petits scripts (escrits en Lua) que analitzen el flux d'events capturats per Sysdig per respondre a determinats events realitzant determinades accions. Per veure els "chisels" disponibles cal fer:

sysdig -cl

Per obtenir informació sobre què fa i quins paràmetres admet un "chisel" determinat, cal escriure:

sysdig -i nomChisel

Per executar un determinat "chisel" cal fer servir el paràmetre -c, així (el "chipsel" *topfiles_bytes* mostra en temps real els fitxers més accedits -llegits i/o escrits- a la màquina).:

sysdig –c topfiles_bytes

Si el "chisel" necessités paràmetres, s'han d'indicar després del seu nom. Per exemple, el "chisel" *spy_ip* necessita la direcció IP de l'extrem del qual es monitoritzarà en temps real els paquets intercanviats amb ell:

sysdig –c spy_ip 192.168.1.157

Es poden executar varis "chisels" a la vegada, si s'escau. També es poden combinar amb filtres, indicats al final, opcionalment entre cometes. Per exemple, si no volem que a la sortida del "chisel" *topfiles_bytes* apareguin els accessos a /dev, podríem fer:

sysdig –c topfiles_bytes "not fd.name contains /dev"

O bé voldríem veure només els arxius més accedits en una determinada carpeta, així:

sysdig –c topfiles_bytes "fd.name contains /root"

O bé voldríem veure només els arxius més accedits per un determinat programa:

sysdig –c topfiles_bytes "proc.name=vi"

O bé voldríem veure només els arxius més accedits per un determinat usuari:

sysdig –c topfiles_bytes "user.name=bob"

Un altre "chisel" interessant és *lsof*, el qual mostra els fitxers actualment oberts i quin procés l'obre. Podríem filtrar només, per exemple, els fitxers de registre de l'Apache (error.log, access.log), així ; noteu les cometes simples dins de les dobles (aquest "chilsel" les necessita):

sysdig –c lsof "'proc.name contains http and fd.name contains log'"

O bé mostrar només la llista de connexions de xarxa establertes per l'usuari "root":

sysdig –c lsof "'fd.type=ipv4 and user.name=root'"

Un "chisel" que emula el comportament de la comanda *ps* és l'anomenat ps. Es pot fer servir així (en aquest cas estaríem veient els processos que estiguin escoltant a la xarxa en ports diferents del 80) ; noteu les cometes simples dins de les dobles (aquest "chilsel" les necessita):

sysdig –c ps "'fd.type=ipv4 and fd.is_server=true and fd.sport!=80'"

O bé mostrar la llista de processos que tenen oberts fitxers dins de la carpeta /etc:
sysdig -c ps "'fd.name contains /etc'"

O bé mostrar tots els processos que no s'anomenin "bash":

sysdig -c ps "'not(proc.name contains bash)'"

Els "chisels" stdin i stdout es poden fer servir per mostra l'entrada i sortida estàndar de les comandes executades a partir de llavors (o de les indicades amb filtres). Per exemple:

sysdig -c stdout "proc.name=cat"

NOTA: Si es vol escriure un "chisel" propi, un bon tutorial per començar es troba aquí:
https://github.com/draios/sysdig/wiki/Writing-a-Sysdig-Chisel,-a-Tutorial .
L'API de referència es troba aquí: https://github.com/draios/sysdig/wiki/sysdig-chisel-api-reference-manual

Finalment, dir que una alternativa integrada dins del kernel (més completa però més difícil que Sysdig) és BPF. Una aplicació d'usuari que la fa servir "per sota" que facilita el seu ús és BCC (https://github.com/iovisor/bcc) . Una altra alternativa és Lttng (http://lttng.org).

1.-Instal.la Sysdig a una màquina virtual qualsevol executant aquest conjunt de comandes:
curl -s https://s3.amazonaws.com/download.draios.com/stable/install-sysdig | sudo bash

Sysdig proporciona una altra comanda anomenada *csysdig* (la qual també ha de ser executada com a root) que ofereix la mateixa funcionalitat que la ja coneguda *sysdig* però en forma de TUI (és a dir, similar a com ho fa per exemple la comanda *htop*). D'aquesta manera es facilita molt l'administració de l'eina ja que la majoria d'opcions estan disponibles visualment dins del terminal.

> NOTA: L´ús de la interfície TUI és possible gràcies a què *csysdig*, *htop* i eines similars estan programades mitjançant una llibreria "gràfica" anomenada "ncurses" (https://invisible-island.net/ncurses)

La informació mostrada per pantalla per *csysdig* és actualitzada cada 2 segons. A la part inferior de la pantalla apareixen una sèrie de botons que es poden usar per accedir a diferents aspectes del programa. Els més interessants són:

*Botó "View" (F2): Mostra un conjunt de "vistes" (amb una petita descripció incorporada), les quals no són més que categories d'elements detectats per Sysdig, com ara "Processes", "System Calls", "Threads", "Containers", "Processes CPU", "Page Faults", "Files" o "Directories" per poder sel.leccionar la desitjada (si s'inicia *csysdig* sense paràmetres automàticament es mostra la "vista" Processes).

*Botó "Filter" (F4) : Permet escollir realitzar un filtre simple de cadena (per defecte) o bé un filtre propi de Sysdig si es pulsa F2

*Botó "Dig" (F6) : Mostra una sortida similar a la de la comanda *sysdig*, sense ornaments

*Botó "Legend" (F7) : Mostra una descripció de les columnes mostrades a la vista activa. Igualment, pulsant el botó "Help" (F1) es pot accedir a la pàgina del manual de *csysdig*. Pulsant per segon cop la mateixa tecla (o bé pulsant ESC) es retorna a la pantalla anterior.
*Botó "Sort" (F9) : Mostra una pantalla on s'ha d'escollir la columna usada per ordenar

*Per sortir cal pulsar la tecla "q".

La comanda *csysdig* també accepta els mateixos paràmetres que *sysdig* i, al final d'ells, un filtre (amb la mateixa sintaxis ja estudiada). Un paràmetre propi de *csysdig* és *-d nº* , el qual serveix per definir el número de milisegons entre actualitzacions de la pantalla. Trobareu més informació sobre *csysdig* a https://sysdig.com/blog/csysdig-explained-visually/

D'altra banda, Sysdig-inspect és una eina complementària que, a partir de les dades emmagatzemades en un fitxer de captura d'events prèviament obtingut per Sysdig, en realitza un estudi estadístic detallat, oferint com a resultat un panell gràfic molt intuïtiu que permet estudiar de forma "forense" l'aparició, comportament i característiques d'aquests diversos events al llarg del temps.

En executar Sysdig-Inspect veuràs que a la finestra apareixen quatre seccions: la barra superior permet tenir en tot moment clar en quina secció d'informació estem (similar al que seria la barra de direccions d'un gestor de fitxers); la barra inferior mostra la longitud temporal de la captura i permet restringir la vista de la informació a un determinat rang; la zona central és la més important perquè és on es mostren en forma de "quadrets" els valors més important de les mètriques, classificades segons àmbits i és on es pot sel.leccionar el "quadret" (o "quadrets") desitjat/s per explorar més profundament la informació en ell continguda; i finalment, la barra de l'esquerra mostra les diferents vistes d'informació ("Fitxers", "Directoris", "Connexions", "Processos", "Errors",etc) que es poden activar per veure-hi de maneres diferents el mateix contingut sel.leccionat a la zona central, a més del botó "I/O Streams (que permet veure el contingut dels fitxers o paquets de xarxa associats a una sel.lecció) i el botó "Syscalls" (que permet veure directament les crides al sistema associades a una sel.lecció). Trobareu més informació sobre Sysdig-inspect a https://sysdig.com/blog/sysdig-inspect-explained-visually/

2.-Crea, a la màquina virtual que estiguis fent servir, un fitxer de captura amb la comanda *sysdig -n 1000 -w hola.scap* . Ara tens dues opcions:

*Si tens entorn gràfic a la màquina virtual on tinguis Sysdig funcionant, descarrega dins d'aquesta màquina virtual el paquet adient (.deb o .rpm, segons sigui la distribució que estiguis fent servir) corresponent al programa Sysdig-inspect (disponible a https://github.com/draios/sysdig-inspect) , i instal.la'l.

*Si no tens entorn gràfic a la màquina virtual on tinguis Sysdig funcionant, descarrega dins del teu sistema Ubuntu real el paquet adient .deb corresponent al programa Sysdig-inspect (disponible a https://github.com/draios/sysdig-inspect), extreu el seu contingut amb un descompressor qualsevol. L'executable del programa el trobaràs dins de l'arxiu "data.tar.gz" (que caldrà descomprimir també), a l'interior de la carpeta "/usr/lib/" amb el nom de "Sysdig Inspect". En aquest darrer cas, però, hauràs d'aconseguir copiar el fitxer de captura obtingut a l'apartat anterior a la màquina real d'alguna manera (el més fàcil, tenint en compte que si has fet els exercicis anteriors ja hauries de tenir un servidor SSH funcionant a la màquina virtual, és executar una comanda similar a aquesta a la màquina real: *scp* usuari@ip.maq.virt.ual*:/ruta/arxiu/hola.scap /ruta/carpeta/real*)

Falco (https://falco.org) is a behavioral activity monitor designed to detect anomalous activity in your applications. It lets you continuously monitor and detect container, application, host, and network activity... all in one place, from one source of data, with one set of rules. It can detect and alert on any behavior that involves making Linux system calls (collected by underlying Sysdig's infrastructure).For example, you can easily detect things like:

*A shell is run inside a container
*A server process spawns a child process of an unexpected type
*Unexpected read of a sensitive file (like /etc/shadow)
*A non-device file is written to /dev
*A standard system binary (like ls) makes an outbound network connection

By default Falco looks for its configuration in two yaml files: "/etc/falco/falco.yaml" and "/etc/falco_rules.yaml" (but their paths can be dynamically configured using the flags: *falco -c <config file> -r <rules file>*). The first file controls several logging and high-level configuration items (for instance, there are several options for configuring security event output -to a file, to syslog, to stdout/stderr or to a piped spawned program (like an email client, for instance)- along with two options for formats -text vs JSON-, etc). The second file serves to define the default rules which will be used to detect specific events about the system behaviour and, consequently, to trigger associated alerts.

NOTA: As we said, "falco_rules.yaml" file contains a predefined set of default rules designed to provide good coverage in a variety of situations; the intent is that this rules file is not modified as it is replace with each new software version. But if you want to add/override/modify these rules you should use the local falco rules file, installed at "/etc/falco/falco_rules.local.yaml" (and empty by default). This file will not be replace with each new software version and it's read after "falco_rules.yaml", so rules defined in "falco_rules.local.yaml" take precedence.

Default Falco rules provided by "falco_rules.yaml" file (and customized rules written in "falco_rules.local.yaml" file by us) can be very different in scope but they all follow the same definition pattern. For instance, following rule is one called "Modify binary dirs":

```
- 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
    pcdline=%proc.pcmdline operation=%evt.type file=%fd.name %evt.args)
  priority: ERROR
```

As you can see, there are a few elements to building a rule. Let's look at these fields in detail:

rule: the identifier of the rule.
desc: a description of the rule, e.g. "rule for alerting on network traffic"
condition: a rule written in the Sysdig filtering language and, optionally, using Falco macros
output: The output message for the rule in Sysdig's output formatting.

priority: Nivell de prioritat de la regla (el seu valor pot ser "debug", "info", "notice", "warning", "error", "critical", "alert" o "emergence"). Si el seu valor és igual o més greu que l'indicat a la línia priority de l'arxiu falco.yaml, la regla associada es tindrà en compte; si no, la regla s'ignorarà

A Falco macro is a name assigned to a set of elements which match a specified condition. For instance, in above rule there was the "bin_dir_rename" macro which is defined at the beginning of "falco_rules.yaml" file like this:

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

There's predefined lists in "falco_rules.yaml" file, too. Its definition begins by – *list:* mark.

NOTA: Teniu molta més informació sobre com definir regles, macros i llistes a la documentació oficial, disponible a https://github.com/falcosecurity/falco/wiki/Falco-Rules

Es pot executar Falco de dues maneres: o bé en forma de dimoni en segon pla (*sudo systemctl start falco*) o bé en primer pla (*sudo falco*). Fes-ho d'aquesta segona manera (per això hauries de tenir el dimoni apagat) i, en un altre terminal diferent, prova de modificar un fitxer que estigui dins de la carpeta /etc. ¿Què veus al terminal on s'està executant Falco? ¿I si ara crees un nou fitxer dins de la carpeta /bin, què veus?. Finalment, atura Falco amb CTRL+C

3.- Instal.la Falco a una màquina virtual qualsevol executant aquest conjunt de comandes:
curl -s https://s3.amazonaws.com/download.draios.com/stable/install-falco | sudo bash